

DOCUMENT RESUME

ED 051 665

24

EN 009 043

AUTHOR

Clutterham, D. R.

TITLE

A Method for Evaluating Student Progress in Undergraduate Computer Science By Use of Automated Problem Sets. Final Report.

INSTITUTION

Florida Inst. of Tech., Melbourne.

SPONS AGENCY

Office of Education (DHEW), Washington, D.C. Bureau of Research.

BUREAU NO

BR-7-D-080

PUB DATE

31 Jan 70

GRANT

OEG-4-8-070080-0015-057

NOTE

121p.

EDRS PRICE
DESCRIPTORS

EDRS Price MF-\$0.65 HC-\$6.58
Automation, Computer Assisted Instruction, Computer Programs, *Computer Science Education, *Grading, *Instructional Aids, Programing, *Simulators, *Student Evaluation

ABSTRACT

As an instructional aid for beginning computer science courses, two systems are described which permit the automatic diagnosing and grading of student prepared problems. The first system, called SIM 610, is based on a simulator which performs by actually running student programs prepared for a simple tutorial computer used in the classroom. The simulator, which will run on any computer with even a limited FORTRAN IV capability, simulates a single address, six decimal machine with 15 basic instructions, nine index registers, and 1000 memory locations. It is capable of taking any problem and a solution prepared by the instructor and using that solution as a standard against which student problems and solutions are automatically compared and graded. The instructor can specify the weighting of factors he considers important in the grading. Diagnostic information is provided to the student on practice runs. A second system, called an Assembly Monitor, provides for the running of student machine language programs on any IBM 1130 computer. It provides a protection system against novice programmers destroying resident programs and, in addition, supplies debugging aids and a grading system very much like that for SIM 610. (JY)

ED051665

BR 7-D-080
PA 24

EM

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE
COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

EM 009 043

ED051665

FINAL REPORT
Project No. 7-D080
Grant No. OEG-4-8-070080-0015-057

U.S. DEPARTMENT OF HEALTH, EDUCATION
& WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED
EXACTLY AS RECEIVED FROM THE PERSON OR
ORGANIZATION ORIGINATING IT. POINTS OF
VIEW OR OPINIONS STATED DO NOT NECES-
SARILY REPRESENT OFFICIAL OFFICE OF EDU-
CATION POSITION OR POLICY.

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE
COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

January 31, 1970

U. S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

FINAL REPORT

Project No. 7-D080
Grant No. OEG-4-8-070080-0015-057

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE
COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

January 31, 1970

The research reported herein was performed pursuant to a grant with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore necessarily represent official Office of Education Position or policy.

U.S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

CONTENTS

SUMMARY	1
INTRODUCTION	2
METHODS	5
Initial Objectives	6
New Direction to Program	8
RESULTS	8
SIM 610 Simulator System	8
Philosophy for Automatic Grading of Student Programs	12
The SIM 610 Computer	14
Problem Definition to the Student	15
Student Programs	19
Initialization of a SIM 610 Program	24
Operation of SIM610	26
ASSEMBLER MONITOR SYSTEM	27
Calling the Assembler Monitor	29
Interpretation of output	33
Operator Procedure and Interpretation of Operator Console Displays	35
Programs, Subroutines and Files	38
Assembler Monitor Use	42
Initialization of Standard Programs	45
CONCLUSION AND RECOMMENDATIONS	
 APPENDIX I	46
Instruction Set	52
APPENDIX II	56
Problem Set	65
APPENDIX III	
Surveys	
APPENDIX IV	
Program Listing	

SUMMARY

As an instructional aid to beginning computer science courses, two systems are described which permit the automatic diagnosing and grading of student prepared problems. The first system is based on a simulator which performs by actually running programs prepared for a simple tutorial computer taught in the classroom. The simulator, which will run on any computer with even a limited FORTRAN IV capability, simulates a single address six decimal machine with 15 basic instructions, 9 index registers, and 1000 memory locations. Several problems which have been used in student classes are given; however, the strength in the system is that it is capable of taking any problem and its solution, provided by the instructor, and using that solution as a standard against which student problems are automatically compared and graded. The instructor can also specify the weighting of factors he considers important in the grading. Diagnostic information is provided to the student on practice runs he makes. The system has been used over four quarters and over 600 students have run problems on the simulator.

A second system provides for the running of student machine language programs on any IBM 1130 computer. This system, called an Assembler Monitor, is programmed in IBM 1130 machine language itself, and will only run on that computer. It provides a protection system against novice programmers destroying resident programs in the machine and, in addition, supplies debugging aids and a grading system very much like that for the simulator.

INTRODUCTION

In the fall of 1965 Florida Institute of Technology introduced an undergraduate degree program in Computer Science. The year 1969 saw the first graduate of this program. In addition to the more than 150 students majoring in Computer Science at Florida Institute of Technology all of the 500 freshmen each year are required to take an introductory course in Computer Science which includes programming. These students pursue degree programs in Electrical Engineering, Physics, Mathematics, and Space Technology.

The Computer Science curriculum at Florida Institute of Technology was designed to present the technology necessary for the undergraduate student to understand computers and their usage and to become a future specialist or generalist in the field. For the non Computer Science major the introductory course taken requires the student to learn programming through actual running of programs he has prepared. For some students this is the only formal training they will receive in programming, but it provides a sufficient basis for their own subsequent work. Others will take additional formal coursework.

Teaching of the quantities of persons taking the computer science introductory course has been a formidable problem for Florida Institute of Technology as well as at other schools in such an endeavor. Since qualified instructors are rather rare there is a natural tendency to load the good ones unmercifully in terms of the number of students they face. In such a situation the instructors find it difficult, if not impossible, to assign and evaluate a representative number of problems. Such is the motivation for a mechanized means of evaluating student problems. A mechanized system also provides for gathering and processing statistical data to assist the instructor in his subsequent problems assignments.

In the process of introducing the unitiated to the use of electronic digital computers, and their programming in particular, a teacher or author is faced with an early decision on the specific computer he uses for illustration. He must either deal with an existing computer or develop an artificial one to demonstrate the characteristics he deems essential. Either approach has advantages and drawbacks.

If an existing computer is taken as the illustration, a dilemma is again faced; either to choose a large machine with an extensive and sophisticated instruction repertoire, or a smaller machine with non essential characteristics imposed on it by short word length. For either case, more complexity is required than is desired to present the rudimentary concepts. The advantage of being able to demonstrate those concepts discussed on an available computer is considerable, however.

Alternatively, if an artificial computer and its instruction repertoire are chosen as the illustrating medium, then a teaching tool can be developed exactly to the author's taste, and need only include essentials, or, it may be embellished as desired. However, the students or readers can never observe the joys of a successfully run program of their own design, or the realistic frustrations of trying to chase down a bug. The results may be like learning to drive an automobile by a correspondence course.

A compromise to the choice between a real and an artificial computer approach is to start from the idealized artificial machine and to simulate its behavior on a real computer. In this way, programs can actually be written for the artificial computer and run (via the real one).

Work done under this contract includes the development of artificial machine language and a simulator on which it runs, and an assembler monitor system which permits ready student access to the use of an actual machine language. The simulator computer is written in FORTRAN and can be used with any computer which has a FORTRAN compiler; the assembler monitor is for use on the IBM 1130 computer only with its machine language. The 1130 computer is in very common use in colleges and other schools and is the Florida Institute of Technology's computer.

The automated problem set undertaken for this contract employs an artificial machine language which is simulated in the universal FORTRAN language so that programs can be written and run to demonstrate the fundamentals of machine language programming. The simulator is designated the SIM 610 (for Simulator of six decimal digit machine). Six decimal digits permits reasonable length data and instruction words. Use of decimal numbers permit the learning of concepts without the added burden of unfamiliar binary numbers, and without numerical conversions which divorce input and output numbers from internal machine

numbers and operations.

The machine language is represented in terms of an instruction set detailed in the report. The pseudo computer of the instruction set has a memory of exactly 1000 words, addressed from 000 to 999 decimal. It has nine index registers referenced by digits 1 to 9. It has a potential for 100 different instructions through instruction codes 00 to 99; however, only 15 of these are used. A computer word length of six decimal digits plus a sign bit (assumed + if not specified) accommodates both single address instruction and data. The 15 instructions fall into categories; data transfer, arithmetic, input/output and branching.

The Assembly Monitor System is designed to permit use of the actual machine language of the IBM-1130 by the student in a controlled environment. This environment permits evaluation of student problems and protects the system itself from being destroyed by student program faults. Since the actual IBM-1130 machine language is rather complicated to use by an apprentice this is considered a necessary feature when assigning students assembly or machine language programs. Such problems are not assigned in the first introductory course which employs the SIM 610 simulator.

METHODS

Initial Objectives

At the outset of the contract the intent with regard to an automated set of problems was the establishment of a continuously reusable set of machine language programming problems. These problems would be of graded sophistication and difficulty and span at least two successive quarters of student experience. An evaluation and grading program was to be developed concurrently which would permit "batch" running of student programs. This program was called the Florida Institute of Technology Student Program Operating Monitor (FITSPOM).

A second task described in the proposal was the preparation of a set of symbolic (assembly) language programs and a means of running these programs in batches and evaluating them also. The intent here was to modify the IBM 1130 Assembler operating under the IBM 1130 Disc Monitor Program, a system available at many schools and colleges.

Both the evaluation programs above were to have data collecting capability on the programs run and were to perform some statistical evaluations on the results. Also both would provide feedback to the student in the form of dumps of his program.

A set of more than 60 machine language problems were developed with optimally programmed solutions and a subset of about 20 of these were picked as a set to be used in the programming courses. The problems were actually used with some of the student classes during initial work on the evaluation program and before it was ready.

A number of unanticipated difficulties arose which necessitated some revisions in the initial objectives. There are described in the following paragraphs.

A major curriculum revision occurred at Florida Institute of Technology affecting all departments and going into effect with the September 1968 term which was in the middle of the period of this grant. In this revised curriculum the courses taken by all students during the first two years are identical and it is not until the Junior year that the differences in the degree programs appear. Such a curriculum has both advantages and disadvantages for both the school and the student. From the standpoint of this grant the advantage is that not only Computer Science students, but all students at

the Institute take an introductory computer course. The disadvantage, from the grant standpoint, is that where the automated problem set was to cover a sequence of courses, it must now cover only a one quarter course and the quantity of problems which can be treated is necessarily fewer. This change did make the requirement for a mechanized handling of student programming problems mandatory for Florida Institute of Technology.

One difficulty which might have been anticipated, but was not originally, was that when the same problems are given to subsequent classes, the optimal solutions also pass along between the students. Thus, any finite set of problems will soon have a complete set of perfect solutions available within the student body so that any student who would rather copy a program than write his own finds no difficulty in doing this. This becomes particularly acute when the course is a mandatory one for all students and does not include just the voluntary Computer Science majors.

With the introductory programming course limited to one quarter its contents had to be very carefully evaluated so that it could best serve the needs of all students - both those Computer Science majors and the larger body some of which would not have any further formal programming. As a result it was deemed necessary to include a higher level language in the course and FORTRAN was chosen. The result is that only about half of the course is devoted to machine/symbolic language. Moreover, the machine language had to be a particularly simple one.

Student problems would really have to be prechecked before running on either the machine language or the symbolic language evaluator because they could fail to run to a finish or worse yet could destroy the evaluator or other resident programs in the computer.

New Direction to Program

As a consequence of the difficulties described, several changes occurred. A very simple machine/symbolic language was developed for an artificial but representative computer. Addressing was done in decimal rather than binary so that concepts could be taught without the additional burden of simultaneous familiarization with another number system. Memory was limited to 1000 words.

The SIM 610 program described in this report simulates this artificial six decimal digit computer in that programs in the artificial language are executed as if the computer was real.

Instead of a formalized set of fixed problems, the approach taken was that any problem (prepared by the instructor or an advanced student, for example) could be used as a master, and the students problems would be graded against that as a standard. Thus there is no final formal set of problems; the student problems are simply made up by each instructor for each course as he needs them. Moreover, it is not assumed that the instructor's program solution is optimal, and it is quite possible for a student grade to be higher than that of the standard provided by the instructor. Flexibility is provided for the instructor to place weighting factors on the various points to be considered in grading, changing them from problem to problem or even at different times for the same problem, depending upon where he wishes emphasis placed. For example, if he is emphasizing program running speed, a high weight can be given in the grade for fast running time as actually measured in terms of actual operations used and their execution times.

The SIM-610 simulator has been used for four quarters and with over 600 students. Surveys of student, instructor, and machine operator observations are included in this report. The Assembler Monitor System has been in informal use and aids in the writing of assembly language programs. The grading portion of the Assembler Monitor System has not been completely debugged, but since it has not had to serve large numbers of students this has not proved a problem.

RESULTS

SIM 610 SIMULATOR SYSTEM

Philosophy for Automatic Grading of Student Programs

In order to grade a student's program, it is necessary to determine its operating characteristics, (i.e. what it does). It is not possible to determine what a program does except by going through it step by step, except in specialized cases. This means either running or simulating the student program. Although it would theoretically be possible to determine other factors about a student's program not determinable simply by running or simulating it, the process involved would be too complex and time consuming to be practical.

There is one major objection to this method, however. If the student programmer makes a minor but crucial mistake anywhere in his program, his grade could be reduced to zero, even though the major part of his program works. This can be handled, however, by giving the student programmer enough debugging aids to allow him to debug his program and re-run it for a better grade. It should be noted that in practice, a computer program, no matter what methods used or how skillfully written, is worth nothing if it does not work. (We will take up the question of partially finished programs again later).

It is, therefore, necessary in order to grade a student program, to actually run it either through simulation or by allowing the execution of the instructions of the program.

If the student program is to be graded, however, the grading program must eventually regain control from the student program. This is no problem if the student's program functions properly and exits normally when finished doing the job. However, if the student's program contains an infinite (unending) loop, the grading program must be able to abort the student program and tell the student the reason for aborting. This can be best done by aborting the student program after a certain amount of run time or after a certain number of instructions have been executed (which-ever is more conveniently available on the system). The maximum amount of time thus set, must be large enough to allow even the inefficient student's program to complete execution; yet not allow the computer to be tied up an excessive amount of time on programs containing infinite loops. As a backup to this, it is sometimes useful to allow the operator to tell the grading program to take control. The specified method or combination of methods must be matched to the computer being used.

It should also be noted that this same instruction

count or runtime can be used later in grading the student program (see below).

It is necessary, therefore, to gain control after the student program is through executing, even if it has an infinite (endless) loop.

When the grading program has gained control, it must determine whether or not the student program has done the job assigned. In some manner the grading program must be told which problem the student is doing. It must also have been given before the student program was run, enough information to determine whether the student did the problem properly.

In order to prevent cheating, all problems should be designed so that the output is a function of the input. For example, a problem to sum the first 100 integers is not a function of an input parameter. Specifically, the answer is a constant, 5050. The problem can be made suitable if the sum of the first "N" integers is required, where "N" is input to the student program. So long as the student does not know what value "N" will be when his program is finally graded, he must do the problem correctly in order to be assured of the correct answer.

In order to be sure that the student will not be able to cheat in this manner, the input data should be changed from practice runs before the final run of the student programs when the grades are recorded for the instructor.

In order to do the above functions, the grader must be able to feed input data to the student program. It must also have the proper answers to the problem based on this input data. The grader must also be told if some of the answers are more important than others.

What, then, should the grader do if the student programmer gets only part of the right answers? Partial credit can be given for some of the answers correct, the answers in the wrong order, or in the wrong places without too much difficulty. It should be remembered, however, that if the students are given sufficient opportunity to debug their programs, there will be little need for the grading routine to have these capabilities.

It is necessary, therefore, for the grading routine to calculate whether or not the student's program did the job required on the basis of his answers being correct for the given input.

Since most students will complete a program that does the job correctly, the students grade must be based upon other factors in addition to the amount of the job completed. The best factors are those actually used to judge practical programs in industry: Runtime (or number of instructions executed in the student program if more easily available), and program length (ie. amount of storage space used by the student program). In addition, if the student program ended for some reason other than normal exit (ie. invalid instruction executed, excessive runtime, or other reason), then credit should be taken off.

The following formula is implemented as a weighting function to calculate the student's raw grade.

$$G = J \times E \times (a / R + b / L + c)$$

where

G = Raw grade to be computed;

J = A factor whose value is zero if no indication was found of the job being done, and is maximum if the job was done completely correctly by the student program;

E = A factor whose value is maximum if the student program ended in normal exit;

R = Runtime (number of instructions executed);

L = Length of the program in core; and a, b, c are positive "weighting" constants for the given problem.

One method of establishing "a", "b", and "c" is to make "a" and "b" functions of the runtime and length (respectively) of a standard program, prepared by a proficient programmer that does the job correctly. This standard program can also be used to initially calculate the proper output from the given input for use by the grader. The constant "c" provides a basis for a non vanishing grade even in the event of vanishingly small credit for runtime, R, and length, L.

Finally, this raw grade must be curved against that of the other students doing the same problem. It is our experience that the raw grade curve can vary widely from one problem to another. Therefore only if the student's raw grade is compared to that of others doing the same problem can his grade be curved properly. All student pro-

grams must be run for a grade before any can be given a grade in familiar letter (A,B,C,D, or F) or percent (100% to 0%) form. The raw grade (based only on the standard program for the problem) can be given each time the student program is run; even for debugging.

The grading program calculates the student's grade on the basis of whether or not he did the job, the number of instructions executed (or the runtime, if available), the length of the program (how much space it uses in core), and how well his program did relative to the other students doing the same job. Moreover, the grade can be weighted by the instructor depending upon where he has placed emphasis in the programming assignment.

Finally, it is necessary to output the information thus determined by the grader. The student is given as much information as necessary. This includes a program listing, reason for exit, runtime, length in core, and whether or not the program has completed the job successfully. In addition, debugging aids such as tracing all or part of the student's program as it executes are included. When the programs are run for the final grade, information is supplied to the instructor so that the grades can be curved and recorded.

The SIM 610 Computer

The SIM 610 is an artificial machine, simulated in the FORTRAN language, which will permit the student to program in machine language, and run as if his program were performing on an actual machine. The simulated computer has a word length of 6 decimal digits plus sign. When words are used for instructions, they are broken into three fields. The first two digits are the operation code, the next digit refers to any one of nine index registers, and the final three digits permit addressing any one of 1,000 addresses. Registers and data flow in the SIM 610 computer, are shown in Figure 1. Following Figure 1, let us trace the operation required for the execution of a single instruction. The instruction address register will contain the address of the next instruction to be executed. Making the assumption that the tag register reads 0 (that is that none of the index registers are referenced) the address from the instruction register passes through the adder with nothing added to it and enters the memory address register. This results in the selected memory contents being placed in the memory data register, and from here it is transferred to the instruction register. While in the instruction register, the first two digits identifying the operation go to operation control to be decoded into the actual operation to be performed. The tag digit goes to the tag select switch. Here one of the index registers is identified if the tag digit is between one and nine. Finally the address is transmitted back to the memory address register through the three digit adder at which time the contents of one of the index registers may be added if it had been previously identified. The number now in the memory address register identifies the location of data in memory and this data is then brought into the memory data register. From the memory data register, the data may pass either to the input-output control, or to the transfer added and accumulator. If the operation is a print, the contents of the memory data register will actually be printed on the output print device of the real computer. If a data transfer operation is involved, such as a load accumulator, the data will pass through the transfer added into the accumulator. If an arithmetic operation is involved, such as subtract from accumulator, or add to index register, the transfer adder will pass the data in the proper direction. Arithmetic operations may cause either the sign latch or the overflow latch to be set. The subsequent use of these latch indicators is described in Appendix I where each of the commands is detailed.

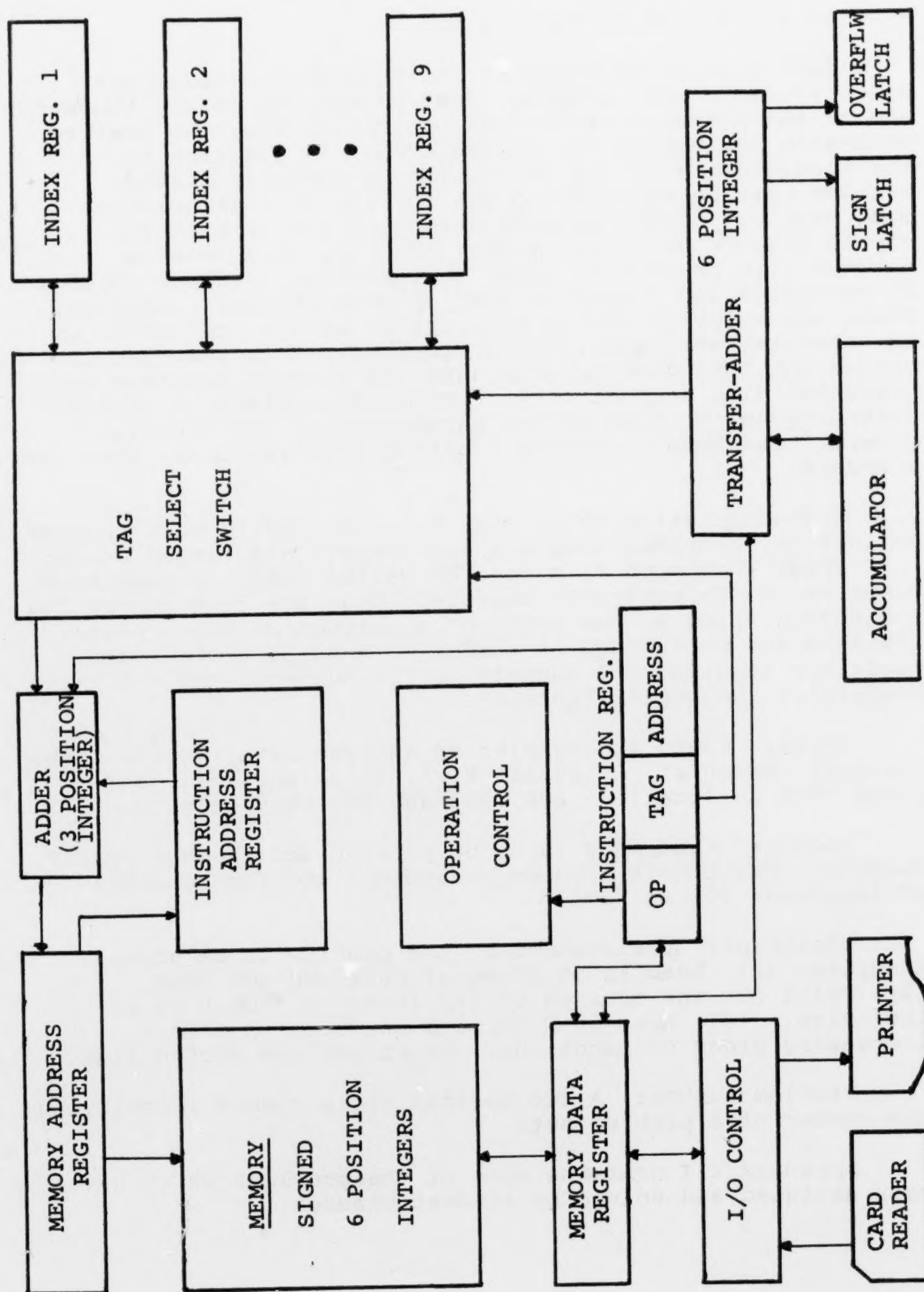


Fig. 1

Problem Definition to the Student

Each problem included in the automated problem set which students must program, must be defined to the student and to the computer simulation program so that the desired automatic evaluation can be achieved. In addition to the fundamentals of the definition, a properly solved problem must be supplied to the computer. This solution must meet all of the specifications of the problem and should also be well programmed; that is, it should be optimum with respect to those characteristics where optimum is specified and should be near optimum in other respects. Thus, the solution should be prepared by the instructor or an otherwise well qualified programmer. This solution is called the "standard program" and all student programs are evaluated with respect to it. Nothing precludes a student bettering one or more of the parameters of the "standard program" and thus receiving a better relative score than the standard.

Characteristics which must be specified in each problem definition, provided they are appropriate are listed below:

Read: How much data must be called into the simulated computer by the program? **Example:** Read one card containing a number N which is the order of a polynomial whose coefficients are on subsequent cards. (A total of $N+2$ data cards are required: 1 containing the number N and $N+1$ containing the coefficients).

Store: Where are results or intermediate results to be located? **Example:** Calculate $f(x)$, $f'(x)$ and $f''(x)$ and place them in locations 200, 201 and 500 respectively.

Output: What data is to be printed and in what order? **Example:** Print N (a problem parameter) and the contents of locations 100 and 101.

Statement: A statement of the problem to be solved. **Examples:** (1) Read in 50 items of data and add them. (2) Print out the squares of the integers from 6 to 20 inclusive. (3) Read in N numbers and sort them in increasing order of magnitude. Print out the sorted list.

Problem number: A two decimal digit number identifying the number of a problem set.

Appendix III contains some of the problems which have been assigned and solved by student classes.

Student Programs

Each student program is submitted as a deck of punched cards as follows: the first physical card in the deck is a beginning of program card, the next cards are the program proper. These are followed by an end of program card and finally by any data cards required. Format of the cards is as follows:

Beginning of program card

Column 1	* (asterisk)
Columns 2-7	000001
Column 8	1 if a deck listing is desired 0 if a deck listing is not desired
Columns 9-13	five digit student number
Column 14-15	two digit problem number
Column 16	(blank)
Columns 17-51	students name (LAST FIRST)

Program card

Column 1	+ or - (blank is treated as +)
Columns 2 and 3	operation code (see Appendix I)
Column 4	tag digit (0 if no index desired) (1-9 for index register)
Columns 5,6,7	three digit address (000-999)

End of program card

Column 1	* (asterisk)
Columns 2-7	999999

Data cards

Column 1	+ or - (blank is treated as +)
Columns 2-7	six digit integer (leading zeros if necessary).

When running programs for practice and debugging, the student should supply his own data deck following the end of data card and use an illegal problem number (e.g. 00). The data he supplies is strictly for his own use, and to satisfy himself that his program is working. If the student has supplied more data cards than required, and the program finishes before using all of them, SIM simply ignores the subsequent cards as it looks for the next students beginning of program card and starts on the next program. If the student has supplied fewer data cards than required and the attempt to read another card brings out the next students beginning of program card, then the present program is terminated and the next one begun. When a program is run for credit, data cards are not supplied by the student and instead "standard" or test data is supplied by the system from disk file storage just as if it were actual cards being read on command.

The first output command executed by a student program starts a new page of printing and prints one word of data from its effective address. Execution of each subsequent output command causes one item of data to be printed on a fresh line. If the trace program is in effect, the output will be intermixed with the trace, but still on a separate line.

Each run of a student program is provided with a trace of the first 25 instructions executed. Trace information (figure 2) includes on one line, the following information:

- XEQNO - the number of the instruction just executed (1-25)
- ADDR - the decimal address of the instruction just executed.
- C(ADDR) - contents of the address above (i.e. the instruction just completed.
- MNEMONIC - monemonic instruction including tag and decimal address.
- C(XR) - contents of index register referenced (before)
- EA - effective address in instruction
- C(EA) - contents of effective address (before)
- C(ACC) - contents of accumulator (before)
- C(XR) - contents of index register referenced (after)
- C(EA) - contents of effective address (after)
- SIGN - sign latch setting
- OVFL - overflow latch setting

Another helpful output from a student's program run is the memory dump. This dump consists of up to 100 lines of printout, each line containing ten words (sign plus six decimal digits). Each line is headed by a decimal identifier indicating the first word of the 10 word block it contains. No blocks (lines) are printed if at least one word in the line was not changed by either writing or executing the program. Unchanged words are left blank in a line. Thus, a few lines of printout may suffice to show everything that changed in a short program. In addition (in fact prior to) the memory dump, the contents of all index registers are printed sequentially on one line. Those which were unused are again left as blank in the printout.

Additional comments which may assist the student in debugging, are provided with the trace and dump and include one of the following;

- EXECUTION COMPLETE
- PROGRAM TERMINATED DUE TO EXCESSIVE RUN TIME
- INVALID INSTRUCTION ENCOUNTERED AT ----
- EXECUTION TERMINATED BY INSTRUCTION AT ---- ATTEMPTING TO READ 1ST CARD OF NEXT PROGRAM INTO ----.

Finally, scoring information is included with calculated scores. On a grading run, the standard program weighted score is shown, otherwise it is zero.

Figure 2 is a SIM 610 diagnostic printout for the student as described in this section.

PROBLEM NO. 12

STANDARD

EXECUTION COMPLETE

RUNTIME		LENGTH OF DECK		NO OF CARDS READ		NO OF ANSWERS WRITTEN		RAW GRADE
YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	
1246	1246	38	38	16	16	15	15	32767
POINTS RECEIVED FOR---		ANS IN CORR LOCATIONS		WRITING ANSWERS		TOTAL		STANDARD
READING DATA	STANDARD	YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	
YOURS	0	435	0	360	0	1045	0	0
250								
SIGN	OVERFLOW	ACCUMULATOR						
0	0	-999910						
10	000000	000015						
20	000000	000015						
30	000000	000015						
40	000000	000015						
200	000456	499999	888889	-000001	000000	000000	-112000	100001
210	000008	102250	000300	-000060	000000	000000	000300	000456
300	-999910	-112000	-000005	-000001	000000	000000	000300	000456
310	100001	102250	499999	888889	000000	000000	000300	000456
400	-999910							
900	000001							

Initialization of a SIM 610 Problem

The "initialize grader" program (INITG) accepts a set of ten (10) cards containing parameters of the problem to be run, and together with other systems programs "load program" (LOADP) and "dump grader" (DUMPG) and "auxiliary initialization program" (INI2G) provides the problem description to the simulator. These ten cards and their content and function are described in the following paragraphs.

Card 0: Character set card

Columns 1 - 16: The integers and operation symbols 0123456789-b+&*b where b designates a blank.

Column 17: Data Set Code (an integer from 1 to 6 inclusive).

The character set identifies the permissible character set and the data set designates a pair of records to be read from "Simulator data" (SIMDT) into DATA1 and DATA2 for use when the standard program executes a read card instruction.

Column 19: Final Grading Indicator. Set to 1 if the points and calculated grade of a student program are to be stored in SMSTU. Not used during initialization.

Cards 1 through 9 are the program description and all have the same format - 10 fields of six place integers, starting in column 1 and having two blanks between fields.

Card 1: Problem number

Field 1: Problem number. This is the record number in the "File of Standard Grades" (FSTDG).

Card 2: Read Groups

Consecutive Fields: Number of cards required in each group (NRDSR) for a number of groups up to 1 and including ten.

Card 3: Read Group Start

Consecutive Fields: The location of the first card in each read group corresponding to card 2 (LOCRD).

Card 4: Store Answers

Fields 1 - 5: Each field gives the first of a sequence of consecutive locations in which the student program is to store answers (LCANS).

Fields 6 - 10: The length (number of answers) of each of the sequences starting in the respective LCANS locations above (NANSR).

Card 5: Points Credit

Consecutive Fields: Each field stores the number of grade points credit to be given for correct answers (data matching the standard problem) for the read groups and their starting locations as given in the respective fields on cards 2 and 3. (PTCR)

Card 6: Proper answer location

Fields 1 - 5: Each field gives the number of points for placing computed answers in the proper locations (regardless of their correctness) as credit for satisfying this part of the problem specification. Proper locations are specified by the corresponding fields 1 - 5 and 6 - 10 on card 4. (PTCA)

Fields 6 - 10: not used.

Card 7: Correct Answers

Fields 1 - 5: Each field contains the number of points to be given for each correct answer found in the locations identified by card 4. (PTCC)

Fields 6 - 10: not used.

Card 8: Printed answer locations

Fields 1 - 5: Each field contains the number of points to be given if the correct answers are found stored in the appropriate group for printing (even if not printed in the correct sequence). (PTCW)

Card 9:

Field 1: Number of points credit if student program execute same number of card read instructions as standard program. Locations where the data read is placed is not considered here. (PTCKN)

- Field 2: Number of points credit for obtaining each correct answer but storing it in an incorrect location (although within total area designated for answer storage). If an essential ingredient of the problem is intended to be sequencing or placement of results then credit points should be set to zero. (PTCO)
- Field 3: Number of points credit for obtaining correct result for output but storing it in an incorrect location (although within the total area designated for output data storage). (PTWO)
- Field 4: Number of answers written by standard problem. This number appears on student's dump but is not given any point value by the system. (NANS)
- Field 5: The contents of this field gives the starting point within the data file for the problem under execution for the reading of simulated data cards as called for by the student (or standard) program. (FDATA)
- Field 6: The number in this field establishes a maximum on the number of operations executed by a given student program. If this many steps are executed, it is assumed that the program is in a loop or is otherwise excessive in its running time and the program will be terminated. (MAXRT)
- Field 7: Percent of grade for run time (steps executed for solution). (PCGRT)
- Field 8: Percent of grade for program length (length of student deck). (PCGPL)
- Fields 9 - 10: not used.

Figure 3 (a and b) illustrate an actual set of cards from a problem set. This may be correlated with problem 3 in Appendix II.

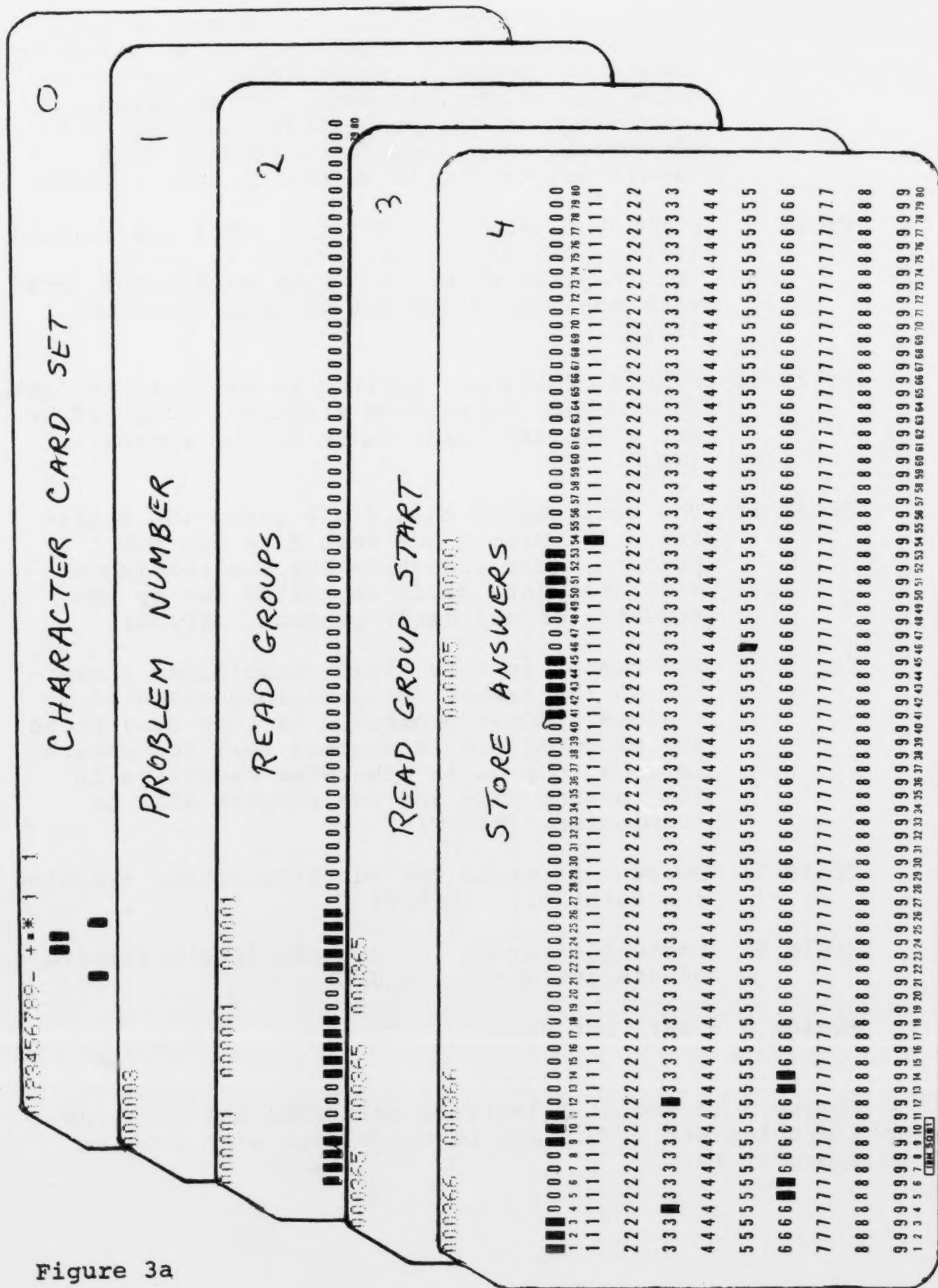
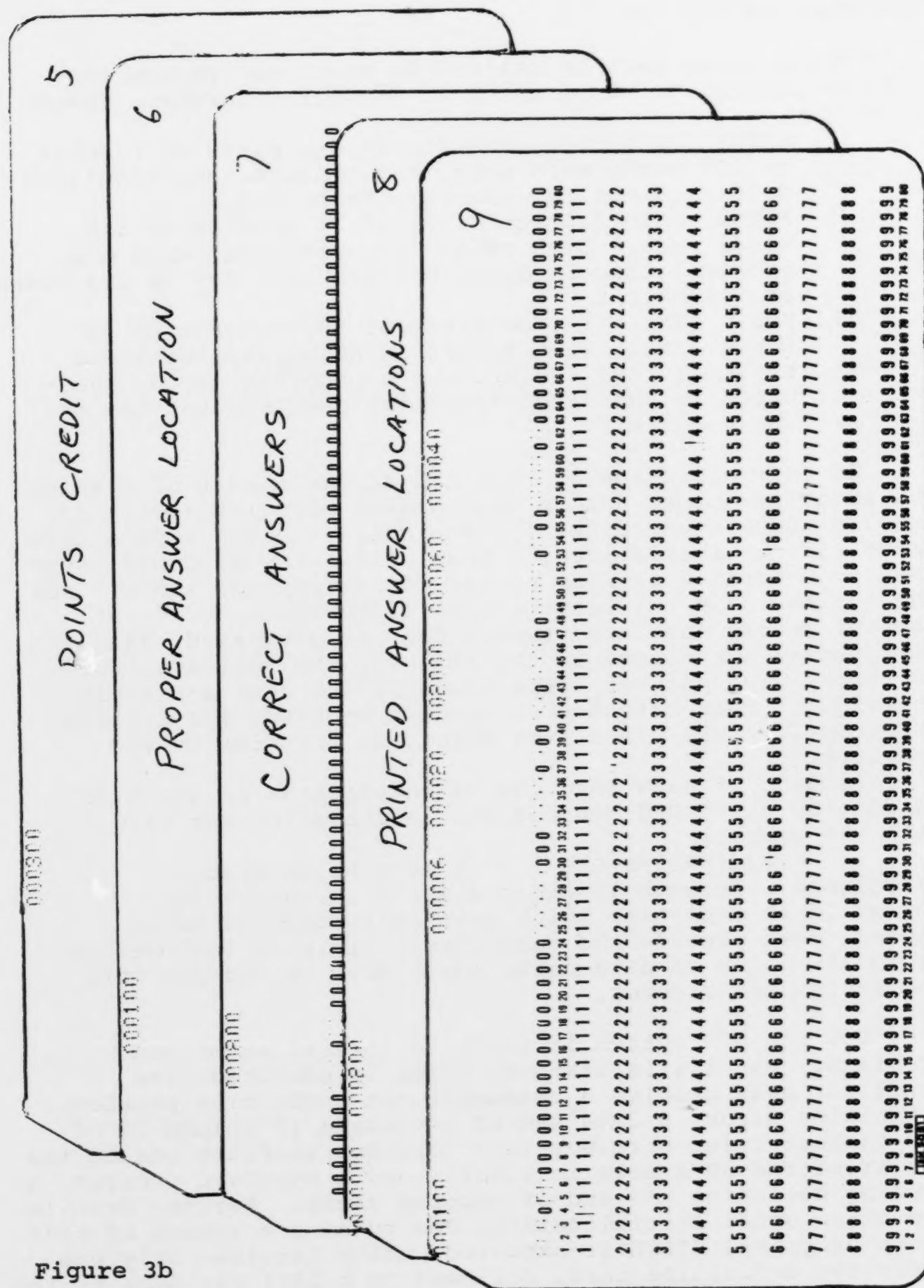


Figure 3a



Operation of SIM 610

Three files must be defined in order to prepare the SIM 610 program to run a batch of student programs. These files are:

- 1) SIMDT This file consists of six pairs of records of 106 words each pair, and contains any simulated data required for problems to be run.
- 2) FSTGD This file consists of 24 records of 160 words each. Each record is associated with one problem; thus 24 different problems may be evaluated in one batch.
- 3) SMSTU This file consists of 800 records of 40 words each. Each record is associated with one student; thus 800 student's programs may be evaluated in one batch (assuming each student has only one program.)

An initialization is required at the outset of a batch run in order to: a) assure that grades associated with any problem numbers undefined for the current batch give a zero grade (if not initialized, a meaningless result could occur when an undefined problem number was attempted) and b) set the "pointer" in the first record of SMSTU to the first student record (as each student deck is processed, data on his program are placed in the SMSTU at the next available position. The pointer keeps track of the next available position so that batching of student problems for grading may actually take place over more than one time on the computer.)

Loading of data into one of the records of the SIMDT is done by the INDFG subprogram. A character set card, with the symbols

0123456789-b+&*b (b is a blank space)
in columns 1 through 16 and a digit 1 through 5 in column 17 to designate which pair of records is to be loaded, must precede the data cards. This is followed by up to 106 pairs of data cards which will be entered into the designated records.

Now the SIM system is ready to initialize or run problems. For initialization, INITG is executed, and reads for each problem a character set card, nine problem definition cards, a data set of 106 cards if column 17 of the character set card was zero (did not indicate one of the six prestored DATA sets in SIMDT), and a standard program including beginning and end of program cards. Further details are given under Initialization. For running a series of students programs, STRTG is executed, which requires only one character definition card, followed by a DATA set deck if the character card so indicates, and then the student programs

stacked one after another. Normally students are given some time to debug their programs, and the results may not be desired to be recorded beyond the listing - dump which is given to the student. This will occur if anything but a 1 is in column 19 of the character set card. When the deadline for finished student programs has passed, STRTG is run using a 1 in column 19, and the student's student number, name, points received (3 categories) and raw grade are stored on a record of SMSTU for each program, except those with mispunched cards (such as a number in column 1), which are not executed or dumped.

Whereas initialization may be terminated at the end of the present program by turning off sense switch 2, no provision is presently made for exiting form SIM 610 in normal operation (under STRTG), since runs are generally of long duration and abnormally terminating a FORTRAN program is simple with most computers.

After a class or group of students programs have been run for grading, the file SMSTU should be dumped to cards for reduction to instructor-useable forms. The program DMPFG accomplished this, and also makes a listing. This gives the programs in the order run, and is useful for finding decks or listings (if not yet returned/given to instructor) or identifying mispunched programs, which are not run, and are in the deck but not on the list. The deck is used in conjunction with a simple listing program and a card sorter, as described below.

The cards may now be sorted in ascending raw grade order and separated by problem, giving a list useful for marking grade divisions; they may be resorted in alphabetical order or student number order for instructor's convenience. An advantage over on-disk sorts arises if correction is desired of cards which do not have last name first, or have other obtainable data missing. If more than one class is represented, the cards may be sorted on the field(s) chosen to distinguish classes, and each class deck listed in various sequences to the taste of the professor. In fact, the separate decks may be given to each instructor to cross index as he wills. Note that the original file is still available until INTFG is used to clear it. (Caution: if same problems are to be reused, references to the file FSTDG should be removed from INTFG, or else this will also be cleared; however, it is normally desired to change to a new data set both before and after grading, thus requiring reinitialization of the problems anyway).

ASSEMBLER MONITOR SYSTEM

The Assembler Monitor System differs from the SIM 610 system in a number of ways. First of all, the Assembler Monitor System uses an actual computer language -- that of the IBM 1130, a small general purpose computer, and thus can only be used on an IBM 1130. As was discussed before, the SIM 610 system can be used on any computer that has a FORTRAN compiler.

There are some advantages to the Assembler Monitor System (AMS), however. Unlike SIM, AMS can use subroutines, including all subroutines available for the system. This also means that AMS allows more flexibility in input/output and allows for problems of much greater complexity.

Two further uses for AMS were found during development. Like most small computers, the IBM 1130 has no memory protect hardware, and no available software to provide this feature. Therefore, we developed as part of AMS a software memory protect to prevent the student from accidentally destroying the Assembler Monitor itself, or the core-resident portion of the IBM supplied monitor-supervisor system. This feature of AMS has proved useful in itself as a debugging aid for the IBM 1130, especially for hard-to-debug assembler language programs and subroutines. Secondly, it was found that AMS could monitor FORTRAN programs on the IBM 1130 just as easily as assembler language programs, thus opening the way to additional uses for the system.

This memory protect software, a necessary part of the Assembler Monitor System, is an extremely complex system in itself. It comprises most of the AM program, which consists of more than 1000 cards. It is written in the assembler language of the IBM 1130.

The portion of this report on the Assembler Monitor System is presented in the form of descriptive handouts to those using the system, and has worked quite effectively. Each of the subsequent sections is such a handout.

Calling the Assembler Monitor System

In order to put your program under the control of the Assembler Monitor System, it is only necessary to call AM from your program, giving it the problem number and your student number. However, there are two pitfalls that must be avoided:

1. Your call to AM must be physically the first CALL or LIBF in your mainline program.
2. Your call to AM should be the first executed statement in your mainline program. Should any instructions be executed before the call to AM, they will not be under control of the Assembler Monitor System.

Your input and output are in COMMON, located at the very end of core. In order to set aside this space at the end of core, you must use a COMMON statement.

You should not attempt to call AM more than once in any given program. An attempt to do so will result in the Assembler Monitor System suppressing further execution.

Below is a sample program including calling sequence for the Assembler Monitor System. (Numbers in next line are card column positions.)

```
1           21       27       35       42
```

```
//JOB
```

```
//ASM
```

```
*COMMON 48
```

```
*LIST
```

(Note: 48 is a sample number, only)
(Note: optional)

Program, constants,
etc.; not including
CALL or LIBF
statements.

```
START CALL  AM
      DC    PRNO
      DC    STNO
```

Program, constants,
calls, libfs, etc.

```
STNO  DC    417
PRNO  DC     4
```

Your student number
Problem number

Program, constants,
calls, libfs, etc.

```
END      START  Last card
      27
```

Before each machine language instruction is executed, AM tests the instruction to determine if its execution would alter the core resident monitor, alter AM itself, make an invalid entry to a subroutine, or an invalid alteration of a subroutine. If its execution would have one of these undesirable effects, further execution of the student's program is suppressed and a link is made to DBUG as explained elsewhere. Further execution of the program is also suppressed if a valid exit is reached, or the run time becomes excessive.

If, on the other hand, AM decides that the instruction should be allowed to execute, the instruction counter is incremented and control is passed to the instruction that was tested.

Immediately after the execution of the instruction, control returns to AM by means of a hardware interrupt. This interrupt results from the machine being in interrupt run (also called trace) mode. AM then tests the next instruction, as before. This procedure of first testing each instruction and then allowing its execution is continued until further execution of the program is suppressed, as described above.

To cause the Assembler Monitor System to monitor your program, you need only call AM at the beginning of your program. When control is passed to AM, it reads the student's input data from the disk, initializes parameters to be used during execution to tell how core has been partitioned for the core load, forces the operator to place the machine in interrupt run mode, and gives control to the testing portion of AM so as to test the first instruction of the student's program.

Interpretation of Output

After the Assembler Monitor has decided that the student's program should not be allowed to execute further, control is passed to DBUG. DBUG moves the paper to the top of the page and prints on the right-hand side the student number, problem number, contents of the accumulator, extension, index, carry, and overflow registers and the floating accumulator. Student number and problem number are given as positive decimal numbers; the accumulator, extension, and index registers are given in hexadecimal; the carry and overflow are given as being "on" or "off"; and the floating accumulator is given in hexadecimal and decimal.

On the left-hand side is printed a core map which gives the starting addresses and lengths of eleven consecutive partitions that make up a core load. The lengths of these partitions vary according to the program(s) in the core load.

The first partition is the Index Register Area, which consists of the first four words of core (i.e., addresses 0, 1, 2, and 3). It is so called because it includes the three index registers, which are in words 1, 2, and 3 in core.

The second partition is the resident monitor, which includes the core resident monitor supplied by IBM (excluding the first four words of core) and the core image header which is located immediately thereafter.

The third partition is the mainline program, which includes everything from the end of the core image header to the beginning of the Assembler Monitor (AM).

The fourth partition is the AMS program, which consists of the program AM, and is the in-core part of the Assembler Monitor System.

The fifth partition is the subroutine area, which includes all subroutines, regardless of type, located between the AMS program and the interrupt level subroutine area.

The sixth partition is the interrupt level subroutine area, which includes all interrupt level subroutines except levels two and four.

The seventh partition is unused core. This partition of core is not used by the core load.

The eighth partition is the LIBF transfer vector, which consists of three words for each library function entry point in the core load.

The ninth partition is the floating accumulator, which consists of six words of core used as an accumulator for floating point arithmetic. There is no floating accumulator if there is no LIBF transfer vector.

The tenth partition of core is the CALL transfer vector, which consists of one word for each CALL entry point in the core load. The CALL transfer vector will sometimes include a dummy word in order to make the floating accumulator begin on an even core boundary.

The eleventh and last partition of core is COMMON, which is located at the very end of core. It is in this partition of core that the input and output occur. COMMON is saved between LINKs by the monitor system; i.e., it is still in core when DEBUG and GROUT are loaded in turn.

On the left-hand edge the starting address and length of each partition are printed in hexadecimal. On the right-hand edge the word ADDR is printed beside that partition in which the effective address of the instruction causing the exit was located. If the exit was not caused by the effective address, the word PREA is printed beside the partition in which the last effective address formed was located.

DEBUG then skips a space and prints the instruction causing the exit and the prior instruction in hexadecimal. To the left it prints the real address (the address of the instruction in core) and the loading address (the address of the instruction relative to the loading point of the mainline, which is the address found on a relocatable assembler mainline listing or a FORTRAN mainline listing).

If the program failed to clear location \$IOCT (/0032 hexadecimal), a line is printed indicating this fact. This error would indicate that an interrupt service subroutine was not incrementing or decrementing \$IOCT properly. Location \$IOCT should be zero if and only if there are no I/O interrupts pending.

A line is then printed giving the reason why the student's program was prevented from further execution, i.e., the reason for exiting. This line is printed in the form:

AMS xx (message giving reason for exit) where xx is the error number. The error numbers are given in the following table:

00 Instruction is located in COMMON.
 01 Instruction is located in CALL transfer vector.
 02 Instruction is located in floating accumulator.
 03 Instruction is located in LIBF transfer vector illegally.
 04 Instruction is located in unused core.
 05 Instruction is located in interrupt level subroutine.
 06 Instruction is located in subroutine area illegally.
 07 Instruction is located in AMS program.
 09 Instruction is located in monitor illegally.
 0A Instruction is located in index register area.
 0C Attempt to alter CALL transfer vector.
 0E Attempt to alter LIBF transfer vector.
 10 Attempt to alter interrupt level subroutine.
 11 Attempt to alter subroutine area from mainline.
 12 Attempt to alter AMS program.
 14 Attempt to alter resident monitor.
 15 Attempt to alter word zero in core.
 1A 64 instructions did irrelevant access of core.
 1B Program terminated due to excessive run time.
 1C Invalid instruction.
 20 Valid exit.

Any other indicators indicate an error in the Assembler Monitor System, and should not occur.

Next, the message ADDRESSES OF LAST *n* INSTRUCTIONS EXECUTED is printed, where *n* is a decimal number with a maximum value of 64 giving the number of addresses listed thereafter. If the program ran for less than or equal to 64 program steps, all the addresses, in the order of execution, will be listed. If the program ran for more than 64 program steps, only the addresses of the last 64 are listed. Both the real and loading addresses are listed in hexadecimal.

If any instructions did an irrelevant access of core (i.e., they did no harm, but did no good, either), then the addresses of these instructions come out in a table titled ADDRESSES OF _____ INSTRUCTIONS LOADING IRRELEVANT DATA where _____ is a hexadecimal number. As above, each address is given both relative to the beginning of core ("REAL") and relative to the beginning of the core load ("LOAD").

In the event that the problem number is zero (or is not the number of a defined problem) no LINK is made to GROUT, the program is not graded, and the only other information printed is the program load length (both in hex and decimal) and the number of instructions executed (both in hex and decimal).

If the problem number is that of a defined problem, then a link is made to GROUT which outputs the student's grade and reasons behind it in three sections titled POINTS FOR CORRECT ANSWERS, ADDITIONAL POINTS FOR OUTPUT, and POINTS FOR PROGRAM EFFICIENCY. The total grade is the product of the total points for each of the three sections (divided by one million to scale it down). The total points for each section is printed after the word total at the bottom of each section and is equal to the sum of the points earned under that section as listed under the right-hand column. The points for each line are calculated from how well the student program did relative to the standard on this point. The total grade is printed beside the message TOTAL GRADE EQUALS at the bottom of the page. The total grade and each of the separate totals should range from zero to one thousand, although it is not impossible to make a grade greater than one thousand.

After printing the total grade, control is returned to the IBM supplied monitor supervisor, which begins looking for the next job.

Operator Procedure and Interpretation of Operator Console Displays

With student program decks in the card reader and the system initialized, the console typewriter will display the following message:

SET MODE SW TO INT RUN

At this time, the operator must set the mode switch (located on the right hand side of the display panel) to "interrupt run" and press the "program start" switch. If "program start" is pressed without first setting the "interrupt run" condition, the above message will be printed again. If the machine is already in the interrupt run mode, the message will not be printed. While in interrupt run mode, the "stop" button will have no effect.

The Assembler Monitor System has a provision for terminating a student due to excessive run time (based on a count of operations executed) and this is done automatically. However, an operator may abort a student program by momentarily placing the bit 11 switch on the console in the up position. In case this does not abort the program and cause an appropriate error message to be printed, then the program is not under Assembler Monitor System control.

If an abort is desired while the machine is in the interrupt run mode and not under control of the Assembler Monitor System, the operator must first take the machine out of interrupt run mode and then press "interrupt request." Alternately, he can first press "interrupt request" which will stop the computer, then change to the run mode and press "program start."

If bit switch 0 is up, the program will stop after each machine language instruction is executed under control of the Assembler Monitor System and display the contents of the Accumulator, Extension and Carry and Overflow status.

Bit switches 14 and 15 are used to control student core dumps and displays to the operator during student program execution under control of the Assembler Monitor System. If bit switch 14 is up and 15 is down, all relevant student core content will be dumped on the printer and the system will pass to the next student program. If bit switch 15 is up, the computer will pause and display a coded error number in the storage buffer register, the

address of the instruction causing the exit in the accumulator, and the effective address of the last instruction employing an effective address in the extension register. Upon restarting, if bit switch 14 is also up, then the relevant student core data will be dumped on the printer. With neither switch 14 or 15 up, no pause or dump occurs.

An override feature is provided which may be used with caution: if bit switch 13 is up after a pause caused by a program exit and switch 15 being up then the Assembler Monitor System will return to the student program.

Programs, Subroutines and Files

Running of the student programs is done under the control of the Assembler Monitor System. This system consists of seven main computer programs, several standard subroutines and four data files described briefly below.

The Assembler Monitor Program (AM) serves as a direct monitor over the running of the student's program, with each instruction performed under monitor control. A debugging aid generator program (DBUGT) prints out a trace and other diagnostic aids to the student from information provided by the AM. The raw grade is calculated by a grading program (GROUT) which calculates the students grade, prints it and records it for the instructor. Program GRINT generates information on which the grade is based from the standard problem supplied. Program INITD initializes data for the grading of each student's problem. For the start of a grading run or for each new problem set, the system is reinitialized with program RINIT which clears the data and grade files. A message input program (MSGIN) loads file a message file with the appropriate messages to be used by the DBUGT program.

Subroutines used in the system include the IBM-supplied Commercial Subroutine Package-Version III, and assembler subroutine for floating binary to decimal (FBTD) and the following special subroutines: FORMT and SHIFT are used by DBUGT to decipher assembler instructions HEXIN converts four alphanumeric characters representing a hexadecimal number into the integer equivalent. HEX and HXOUT convert an integer back to hexadecimal. DCOUT converts an integer into five alpha characters representing a number in decimal. OUT prints a line of alphanumeric characters and clears the output buffer to blanks. DSCTR dumps a 320 word core sector (length of one disk sector) in hexadecimal to the printer.

SAVGR contains three records of 320 words per record. Since each disk sector contains 320 words, this file uses three sectors. The initial contents of SAVGR are unimportant because AM loads the file with new data with each new student program. The actual instructions, variables, and constants of AM are stored by that program in three blocks. The three records are the 320 words following respectively the three DSA statements labeled IOAR1, IOAR2, and IOAR3. It is the task of program DBUGT to extract the pertinent data from irrelevant coding. SAVGR is referred to in all programs by symbolic file number 1.

MSGBF also contains three records of 320 words per record, giving three disk sectors. It is used by program DBUGT to print all words interpreting the output of AM including all headings and in converting all numbers from integer format to alpha characters. To initialize MSGBF, program MSGIN is executed, reading data from twelve cards in FORMAT (80A1), and storing the contents on disk. Refer to program listing of MSGIN for contents of data cards. MSGBF is referred to in all programs by symbolic file number 2.

The records of GFILE each contain 16 words with one record generated per student program run under the system for grading. The length of GFILE can therefore be varied with the needs of the user by simply changing the number of sectors specified when the file is set up and by changing the number of records in the DEFINE FILE statement in program GROUT. For example, if the user desired 400 records at 20 words per sector, this would require 20 sectors of disk. The contents of each record of GFILE will be listed and explained later. The contents of GFILE is initially set to zeroes by program RINIT. GFILE is referred to by symbolic file number 3.

DATFT contains ten records of 320 words apiece, giving 10 disk sectors. Each record contains information used by the system in grading a problem of the standard data set. The system can therefore handle a problem set of 10 problems. The corresponding record of DATFT must be reset to zeroes before entering a new standard problem in the problem set. To reset DATFT and/or GFILE, execute program RINIT, following it by one data card of FORMAT (10I2,10x,I2). The first 10 fields indicate which records of DATFT are to be reinitialized. If GFILE is also to be 32 is to be left blank. DATFT will be referred to by symbolic file number 4.

To define these four files on disk, the computer should be given instructions corresponding to these:

```
// JOB
// DUP
*STOREDATA  WS FX SAVGR0003

*STOREDATA  WS FX MSGBF0003

*STOREDATA  WS FX GFILE0020

*STOREDATA  WS FX DATFT0010
```

Since programs DBUGT, GRINP, and GROUT are executed by links and have quite lengthy core-loads, the running of a student program under the Assembler Monitor System can be quite time-consuming. If the user has sufficient area on disk, it is suggested that these programs be stored Core-Image. This will considerably speed the operation of the system. All four data files must therefore be stored in Fixed Area on disk.

The next step is to execute program MSGIN which will read 12 cards of alphanumeric data and initialize file MSGBF (see program listing). This file will be used to generate headers and output information by program DBUGT.

Assembler Monitor Use

The Assembler Monitor System has provision for up to 120 words of input data read by the student program determining the grade on up to 120 words of output. The input is loaded by AM into COMMON, beginning with the last word of core. AM will not load input data beyond the end of a student's specified COMMON. Any COMMON beyond the number of words of input is filled with zero or some other easily recognizable "garbage word" specified by the instructor. This is done as a debugging aid so that the student can determine by examining a core dump what, if anything, his program has changed. The output must also be in COMMON and within the last 320 words of core. The 120 words of output can be divided into as many as 10 blocks of consecutive core locations and these blocks can be located anywhere within COMMON. This permits freedom to:

1. Give more important answers more credit for grade.
2. Count part of the grade on intermediate answers arrived at in the process of generating the final answers.
3. Remove points for destroying the input in the process of obtaining an answer. A further option is provided to give points for partially correct answers, that is answers either in the correct blocks but in incorrect order, or answers found anywhere within COMMON. This option can be used as a debugging aid by pointing out to the student that he has made only a small logic error in addressing and not written a program that does nothing.

Program efficiency is determined on the basis of five parameters: mainline program length, subroutine length, length of COMMON, number of instructions executed, and a standard curve or bias. The curve is based on the theory that with the high speed of this computer, the length of most programs run under the system, the difficulty of writing in assembler, and inexperience in programming of most students using the system, that a program that works should not receive a failing grade no matter how inefficient it is.

In order to initialize DATFT with the standard input data, output buffer locations, and grading factors the instructor must perform the following operations: First Store subroutines INITD, HEX and HEXIN on disk. HEXIN is

used to translate core addresses entered in hexadecimal (four characters) into integer constants. HEX is used to translate DATFT to hexadecimal characters for dump to printer. INITD takes parameters problem number and standard input and data cards for output locations and grading points and puts them on disk. Since INITD is a subroutine, it cannot initialize its own IO. This must be done by a short calling program (written in FORTRAN). This program must initialize ISS routines for disk, card reader, and line printer and must tell INITD where to find DATFT on disk. For Example;

```
//JOB
//FØR
*ØNE WØRD INTEGERS
*EXTENDED PRECISIØN
*IØCS(DISK,CARD, 1403 PRINTER)
.
.
.
DEFINE FILE 4(10,320,U,K)
.
.
.
program (see below)
.
.
.
CALL INITD(... )
CALL EXIT
END
//XEQ 01
*FILES(4,DATFT)
.
.
.
5 Data cards.
```

The following four integer calling arguments should be passed to INITD if called by FØRTRAN:

1. Problem Number(PROBN).
2. Standard Input (STDIP), the first element of an array up to 120 words long.
3. Standard Input Length (STDIL), the number of words of input.

4. "Garbage" Word (GBGWD), filler for remaining student COMMON; e.g., CALL INITD(PROBN,STDIP,STDIL,GBGWD).

The array STDIP can be initialized by data statements, arithmetic assignment statements, or read statements in integer or A1 format. (Do not use the commercial CALL READ.) If it is desired to place real numbers into STDIP it must be remembered that one extended precision real number fills three words of core and that the first element of a real array should be equivalenced to the third element of the corresponding integer array. This is because FORTRAN arrays are stored in reverse order in core. For the same reason, the first element of STDIP will be placed by AM into the last word in core, and following elements will be stored into descending core locations.

If greater versatility of input is desired, the FORTRAN program can call an assembler subroutine which generates STDIP and in turn calls INITD. In this way, the student can be provided with input in the format of actual instructions, characters in card-code, paper tape, etc. These changes in the calling sequence must be noted: All calling arguments must be addresses of the parameters, not the parameters themselves. Also, STDIP is the address of the last location of input. For example:

	ENT	DATA
DATA	...	
	CALL	INITD
	DC	PROBN
	DC	STDIP
	DC	STDIL
	DC	GBGWD
	EXIT	
PROBN	DC	1
STDIP	BES E	120
STDIL	DC	120
GBGWD	DC	/EEEE
	END	

In this way, STDIP can be filled by such assembler pseudo-ops as:

DC	/...	hex constant
DEC		2-word decimal integer or real constant
XFLC		extended precision real constant
EBC		extended BCD interchange code characters
DMES		printer hex (console, 1132 or 1403)
DN		name code constant.

The instructor can provide, by an LIBF to ZIPCO, paper tape or card-code characters.

Output locations and grading parameters are entered as data on five cards after the //XEQ and *FILES cards (and also after any data cards read by the mainline). The first two cards contain respectively the beginning and ending addresses of up to ten output buffers GROUT is to search for answers. The addresses are to be expressed in four digit hexadecimal, absolute, with two spaces between address, up to ten addresses per card.

FORMAT (10(4A1,2X))

Card three contains five numbers which are the percentage points to be assigned for program efficiency. The first number is for mainline program length, the second for subroutine length, the third for length of COMMON, the fourth for number of instructions executed, and the fifth is the curve. The sum of all five parameters should equal 100. Each number should be expressed as three digits with two spaces between each.

FORMAT (5(13,2X))

Card four contains up to ten percentage points for answers in correct locations, one corresponding to each answer buffer defined in cards one and two. Card five contains three percentage points determining value of partially correct answers. The first parameter is percentage for completely correct answers, the second for answers within the correct buffers but not necessarily in correct order, and the third is for answers anywhere within COMMON. The sum of cards four and five must each equal 100. The formats are the same as for card three. For example (for a machine with 8K core):

```
1FF0 1FD0 1FEO 1FE8
1FFF 1FDF 1FE7 1FE8
```

```
010 010 010 040 030
015 015 040 030
070 020 010
```

At the end of execution, INITD will give a hex dump of DATFT to the printer. The standard input buffer is stored in DATFT in reverse order to that in which it is loaded into core. The first element of DATFT (last element in the FORTRAN dump) is loaded into the last location of core and so forth.

Initilization of Standard Programs

The final step in preparing the system for grading student programs is to run the standard programs. These are to be run in the same manner as student programs, with the following changes in operating procedure:

1. Parameters to be passed to AM are the address of problem number and a student number of -1 (FFFF in hexadecimal).

2. All data switches on the console must be placed in the up position (FFFF hexadecimal).

3. The program will stop after the first instruction with an exit code of 301C hexadecimal in the SBR. All switches except 13 should be placed in the down position and the program started. The program should now stop with 3020 in the SBR (normal Exit). (If a core-dump is desired, put switch 14 up.) Restart the machine. AM will now store the information it has compiled on SAVGR to DBUGT, which will read SAVGR, MSGBF, and DATFT. DBUGT will determine that the program is a standard and will link to GRINP. GRINP will complete the initialization of DATFT with standard output and standard program efficiency. COMMON and DATFT will be dumped to the printer in hexadecimal. A link will be performed back to DBUGT, which will then handle the standard as if it were a normal student program (as a cross-check on the standard.) The standard program will receive a grade of 1000 points. All student programs will be graded in comparison to this standard grade. Student programs can now be run and graded on the system for all problems on which the standard has been initialized.

Computation of Grade

The computation of the student's grade is based on these factors:

- I. Answers
 - A. completely correct
 - B. partially correct
- II. Program efficiency
 - A. Mainline length
 - B. Subroutine length
 - C. Length of COMMON
 - D. Number of instructions executed
 - E. Standard curve
- III. Correct termination of program (EXIT)

To compute I,A, GROUT compares the contents of the output data blocks in the students COMMON to the corresponding standard output block, and computes the ratio of the number

of correct answers the student finds to the length of the block (standard number of correct answers). This ratio is multiplied by the corresponding grading parameter for correct answers (entered into DATFT by INITD, data card #4). The sum of these 10 products is then multiplied by the grading factor for totally correct answers (INITD, first number, data card #5). GROUT then searches the student's output buffers, counting the number of correct answers placed anywhere within the correct data block. The ratio of the number of answers so found to the total number of possible answers, is multiplied by the grading parameter for answers within the correct data blocks (INITD, second number, card #5). All of COMMON is then searched for the correct answers found in any locations, the ratio to total answers is computed and multiplied by the parameter for answers within COMMON. The total points for answers is the sum of points for correct answers, answers within the correct buffers, and answers anywhere in COMMON.

Points for program efficiency are computed as the sum of points for program length, subroutine length, length of COMMON, number of instructions executed and standard curve. Points for program length are computed as the ratio of Standard program length to student program length, times the grading parameter for program length (INITD, first number, card #3). If the student did not receive a perfect score on answers and his program length was less than that of the standard, points for program length is computed as if his program length was the same as that of the standard. Points for subroutine, COMMON, and number of instructions are computed in a like manner.

Total grade is computed by multiplying points for answers by points for program efficiency. 25% of the grade is lost if the program is terminated by anything but a standard exit (AMS 20). A message to this effect is printed. The final grade is then scaled on a factor of 1000. It is important to note that the grade given by the system is based upon a comparison between the student program and a "standard" program, and not between the student and other student programs. For this reason, the final scaling of grades must be left to the instructor. The system does, however, give a fair grade in that the grade is proportional to the worth of the program (if the grading parameters are assigned properly), and that the instructor can easily tell from the output supplied to him, where to scale the grades.

Output of GROUT to GFILE

GROUT supplies certain pertinent information about the student's grade to the instructor by entering a 16 word record on GFILE for each program graded, unless the student passes a negative student number to AM. The

contents of GFILE is as follows:

1. Record number (first record has total number of records saved).
2. Student number.
3. Problem number.
4. AMS exit code.
5. Total grade.
6. Points for completely correct answers.
7. Points for all answers.
8. Points for program efficiency.
9. Program length.
10. Subroutine length.
11. Length of COMMON.
12. and 13. Number of instructions executed. Since a program can possibly execute more than 32,767 instructions, (the greatest possible integer the machine can hold), AM divides the instruction count into two words. The first is the number of instructions divided by 10000, and the second is the remainder of the instruction count. In other words, 13 is the low order four decimal digits and 12 is the upper decimal digits.
14. Number of answers in correct locations.
15. Number of answers within correct data blocks.
16. Number of answers anywhere within COMMON.

CONCLUSION AND RECOMMENDATIONS

Difficulty with a fixed problem set to be used repeatedly, led to the approach employed which permits new problems to be introduced as frequently as necessary. This has been effective over several quarters. Experience has shown that a first program for the student should be extremely simple - something like reading a number into the computer and printing it out. This divorces the mechanics of basic input and output from other programming complexities and gives the student the satisfaction of having been on the computer very early in the course.

Additional instructions have been considered for the repertoire of the simulator. These might include arithmetic and cyclic shifts, multiplication and perhaps even division. Although these would permit the solution of more sophisticated problems and may make the simulated computer more like an actual one, they would not make a major advance to the learning obtained via the current basic machine commands.

Provision is made in the present systems for accommodating the five decimal digit student identification number at Florida Institute of Technology. This is inadequate for some schools and will ultimately be inadequate at F.I.T. when a change to Social Security numbers as identification occurs, as it most surely will.

The Assembly Monitor system is only serving a small quantity of people - those computer science majors who use it in machine language programming. However, they are not required to use it. Moreover, nearly all problems at the machine language level, have been individually designed and must result in a working program. Further work on this program is not recommended at this time.

APPENDIX I

10 T AAA LOAD ACCUMULATOR - LDA

The contents of the Accumulator are replaced by the contents of the effective address. The contents of the effective address are not changed. The Sign latch is set equal to the sign of the contents of effective address. The Overflow latch is not affected.

$EA = AAA + \text{contents of } T \text{ (if } T=0, EA=AAA)$

Example: 10 4 625 EA=625+213=838

Before execution:	After execution:
Accumulator ????????	Accumulator +991246
I/R 4 +000213	I/R 4 +000213
Location 838 +991246	Location 838 +991246
Sign Latch ?	Sign Latch Positive

11 T AAA STORE ACCUMULATOR - STA

The contents of the effective address are replaced by the contents of the Accumulator. The contents of the Accumulator are not changed. The Sign latch is set equal to the contents of the Accumulator. The Overflow latch is not affected.

$EA = AAA + \text{contents of } T \text{ (if } T=0, EA=AAA)$

Example: 11 0 001 EA=001

Before execution:	After execution:
Accumulator -999999	Accumulator -999999
Location 001 ????????	Location 001 -999999
Sign Latch ?	Sign Latch Negative

40 T AAA LOAD INDEX REGISTER - LDX

The contents of the specified Index Register T are replaced by the contents of the effective address AAA. The contents of the effective address are not affected. The Sign latch is set equal to the sign of the contents of the effective address. The Overflow latch is not affected.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 40 9 123 EA=123

Before execution:		After execution:	
I/R 9	???????	I/R 9	-999995
Location 123	-999995	Location 123	-999995
Sign Latch	?	Sign Latch	Negative

20 T AAA ADD TO ACCUMULATOR - ADD

The contents of the effective address are algebraically added to the contents of the Accumulator. The contents of the effective address are not changed. The sign latch is set equal to the sign of the result in the Accumulator. The Overflow latch is set on if sum exceeds +999999 or is less than -999999. When Overflow occurs, high-order digits are truncated. The Overflow latch is set OFF if overflow did not occur.

EA = AAA + contents of T (if T=0, EA=AAA)

Examples:			Over-	Sign
Accumulator Before	EA	Accumulator After	flow	
-999999	-000001	000000	ON	0
-001001	+000001	-001000	OFF	Neg.
-999999	+999999	000000	OFF	0
+010010	000000	+010010	OFF	+
+999999	+000001	000000	ON	0
+999999	+999999	+999998	ON	+

21 T AAA SUBTRACT FROM ACCUMULATOR - SUB

The contents of the effective address are algebraically subtracted from the contents of the Accumulator. The contents of the effective address are not changed. The sign latch is set equal to the sign of the result in the Accumulator. The Overflow latch is set on if the result is greater than +999999 or less than -999999. When overflow occurs, high-order digits are truncated. The Overflow latch is set off if overflow did not occur.

EA = AAA + contents of T (if T=0, EA=AAA)

Examples:			Over-	Sign
Accumulator Before	EA	Accumulator After	flow	
-999999	-999999	000000	OFF	0
-999999	+000001	000000	ON	0
+000001	+999000	-998999	OFF	Neg.
+999998	+000001	+999997	OFF	+
+999999	-999999	+999998	ON	+

42 T AAA ADD TO INDEX REGISTER - MDX

The contents of the effective address AAA are algebraically added to the contents of the specified Index Register T. The contents of the effective address are not changed. The Sign latch is set equal to the sign of the result in the Index Register. The Overflow latch is set on if sum exceeds +999999 or is less than -999999. When overflow occurs, high-order digits are truncated. The Overflow latch is set off if overflow did not occur.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 42 5 002

Before execution:		After execution:	
I/R 5	-999999	IR/5	000000
Location 002	-000001	Location 002	-000001
Sign latch	?	Sign latch	0
Overflow latch	?	Overflow latch	ON

41 T AAA STORE INDEX REGISTER - STX

The contents of the effective address AAA are replaced by the contents of the specified Index Register T. The contents of Index Register T are not affected. The Sign latch is set equal to the sign of the contents of Index Register T. The Overflow latch is not affected.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 41 1 402 EA=402

Before execution:
I/R 1 000000
Location 402 ??????
Sign Latch ?

After execution:
I/R 1 000000
Location 402 000000
Sign Latch 0

60 T AAA READ A CARD - IN

Data is read in from a card and temporarily held in a buffer area. The data in the buffer is then checked for validity. If the first column contains an asterisk, the current program is terminated. If not, the first column must be a blank, plus sign, or minus sign. Blank is treated as a plus sign. Columns 2 through 7 must contain digits from 0 to 9 --- blanks are not allowed. Columns 8 - 80 may contain comments.

If the validity checking does not detect an error, the data is loaded into the core location specified by the effective address. If the data is invalid, the contents of the effective address are not altered. The Overflow and Sign latches are not affected in any case.

EA = AAA + contents of T (if T=0, EA=AAA)

Example: 60 1 427 (data in card, +426351) EA=427+111=538

Before execution:
I/R 1 +000111
Location 538 ??????

After execution:
I/R 1 +000111
Location 538 +426351

61 T AAA WRITE - OUT

The contents of the effective address is printed on the printer, and the paper is advanced one space. The Sign and Overflow latches are not affected.

77 0 000 STOP - HLT

Execution is terminated. The Sign and Overflow latches are not affected. Core is dumped onto the printer, ten locations per line for any line containing a word in which any change has been made in storage during execution.

50 T AAA BRANCH (Unconditional) - B

Control is transferred to the instruction at the effective address. The Sign and Overflow latches are not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

Example:	<u>Core location</u>	<u>Contents</u>
	042	500862
	043	??????
	...	
	862	210044

Execution of the Branch instruction at location 042 will cause the next instruction executed to be the subtract instruction at location 862.

51 T AAA BRANCH NEGATIVE - BN

This instruction causes a branch to the effective address if the Sign latch is Negative. If the Sign latch is not negative, control goes to the next sequential address. The sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

52 T AAA BRANCH ZERO - BZ

This instruction causes a branch to the effective address if the Sign latch is zero. Otherwise, control goes to the next sequential address. The Sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

53 T AAA BRANCH POSITIVE - BP

This instruction causes a branch to the effective address if the Sign latch is positive. Otherwise, control goes to the next sequential address. The Sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

54 T AAA BRANCH OVERFLOW - BØ

This instruction causes a branch to the effective address if the Overflow latch is ON. Otherwise, control goes to the next sequential address. If branch occurs, then the Overflow latch is reset to OFF. The Sign latch is not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

APPENDIX II

Problems 1, 3, 4, 5, and 6 are from the winter quarter 1969. Problems 11, 12, 13, 14 are from the spring quarter 1969.

PROBLEM NO. 1

Given: A set of 100 data cards containing values X_i such that:

$$i = 1, 2, 3, \dots, 100$$
$$-999999 \leq X_i \leq +999999$$

Write a machine language problem beginning in location 0 (zero) to solve the following equation:

$$\sum_{i=1}^{100} X_i \text{ where } 0 \leq X_i \leq 1000$$

i.e.; omit values of X_i outside of the above range from the sum.

Be as efficient as possible.

Write out the answer on the printer.

Store your answer in location 900.

Read the given input data into locations 500-599.

Use index register(s) and conditional instruction(s).

PROBLEM NO. 3

Given: A set of 100 data cards containing values X_i such that:

$$i=1, 2, 3, \dots, 100 \\ -999999 \leq X_i \leq +999999$$

Write a machine language program beginning in location 0 (zero) to solve the following equations:

$$\text{Sum 1} = \sum_{i=1}^{99} X_i \quad (\text{Sum the contents of only the odd numbered locations: } i=1, 3, 5, \dots, 99)$$

$$\text{Sum 2} = \sum_{i=2}^{100} X_i \quad (\text{Sum the contents of only the even numbered locations: } i=2, 4, 6, \dots, 100)$$

Write out both answers on the printer.

Store the answers: Sum 1 in location 900
Sum 2 in location 901

Read the given input data into locations 500-599.

Assume no overflow will occur.

Use any instructions you think necessary.

Be as efficient as possible.

PROBLEM NO. 4

Given: A set of 100 data cards containing values X_i such that:

$$i=1, 2, 3, \dots, 100 \\ -999999 \leq X_i \leq +999999$$

Write a machine language program beginning in location 0 (zero) to perform the following:

- (a) Find ANS. 1 = total number of negative items in the list
- (b) Find ANS. 2 = total number of zero items in the list.
- (c) Find ANS. 3 = total number of positive items in the list.

Problem No. 4 (cont'd)

Read the given data into locations 500-599
Store the answers: Ans. 1 in loc 900
 Ans. 2 in loc 901
 Ans. 3 in loc 903

Use any instructions you think necessary.
Be as efficient as possible.

PROBLEM NO. 5

Given: Two sets of 50 data cards containing values

$$\left. \begin{matrix} X_i \\ Y_i \end{matrix} \right\} \quad i=1, 2, 3, \dots 50$$

such that $-5000 \leq X_i \leq +5000$
 $-5000 \leq Y_i \leq +5000$

Find the sum of the differences $(X_i - Y_i)$ by the following formula:

$$\sum_{i=1}^{50} (X_i - Y_i)$$

Read the first set of fifty cards into locations 500-549.
Read the second set of fifty cards into locations 550-599.
Write out the answer on the printer.
Store the answer in location 900.
Use any instructions you think necessary.
Be as efficient as possible.

PROBLEM NO. 6

Determine and print the first N numbers of the "FIBBONACCI" series. In the "FIBBONACCI" series each number is the sum of the previous two numbers with the first two numbers of the series being 0 and 1.

Example of the "FIBBONACCI" series:
0, 1, 1, 2, 3, 5, 8....(to N terms of the series)

Read the value of N into location 500. Store the terms of the series starting in location 900. Print the terms of the series.

PROBLEM NO. 11

Write a program which will evaluate

$$f(x) = 3x^2 + 2x + 7$$

for x an integer ($0 < x < 100$) to be read in from a data card. Test x after reading to make sure it is correct. Print out the value of x and f(x). Store f(x) in 900. If the value of x is out of the allowable range, print out the actual value of x, 000000 for f(x), and stop.

PROBLEM NO. 12

Write a program which will read (1) a card with the integer $0 < N < 100$. (2) N data cards into N successive locations, then sort the N numbers into ascending order and print them out. Read the data cards into locations 200ff and sort into locations 300ff.

PROBLEM NO. 13

Write a program to read in 25 numbers. These are to be stored in consecutive locations starting at 200. The numbers represent consecutive elements in consecutive rows of a matrix. Perform the transpose of the matrix so that rows and columns are interchanged. Print out the transposed matrix. Store transpose in locations 300ff.

PROBLEM NO. 14

Given three sets of data cards of $N \leq 30$ cards each:
Read the first set of N cards into locations 100, 103, 106,...

Read the second set of N cards into locations 101, 104, 107,...

Read the third set of N cards into locations 102, 105, 108,...

Print out in order locations 100, 102, etc.

N is on first card. (A total of $3N+1$ cards will be read.)

APPENDIX III

This appendix contains summaries of the results of three surveys conducted after the automated problem sets had been used by several classes.

First is the student response to a questionnaire which followed the course.

Second is the concensus of the instructor who taught the course.

Third is observations of the IBM-1130 operator who actually accepted the students programs and batch processed them.

Student Survey on Automated Problem Sets

A questionnaire (Table I) was prepared to ascertain the effectivity of the automated problem sets from the standpoint of the students. This questionnaire and the summarized responses from 134 students are shown. The questions were designed to determine the extent of ease or difficulty which the new (to the students) concept of machine language was assimilated. Results were obtained after the student had subsequently been exposed to, and had written programs in, a compiler language, namely, FORTRAN.

The final question requesting comments on improvement of the course elicited response from approximately fifty percent of the questionnaires. It opened a Pandora's box with a great diversity of opinions expressed. At the extremes, these ranged from the ideas that machine language was a complete waste of time and all programming training should be concentrated on FORTRAN to the desire to have the full quarter devoted to binary machine language with more emphasis on arithmetic and control unit organization. Specific comments also dealt with insufficient demonstration on keypunch, need to have first programs examined in detail by instructor before attempting to run, need for monitors to be better versed in the simulation language and in the problems assigned that quarter. A majority of the opinions expressed reflected the students' personal desires in results of such a course and in their success or frustrations in achieving these desires.

The following numbered observations correspond to the questions of the same number shown in Table I.

1. Less than two percent of the students had any prior experience with machine language.
2. Eighty percent believed that the instruction set was about the right complexity with the rest equally divided between too simple and too complex.
3. Responses were equally divided between those accepting the set as adequate and those desiring a multiply instruction. The fact that a negligible number thought shifting should be included probably indicates that its use was not pointed out to the students.
4. A negligible number of responses felt that the number of branch instructions was excessive and about a third wanted even more variations.

5. Opinion was about 7-5 in favor of a less restrictive I/O set.
6. Opinion was about equally divided for and against inclusion of logical instructions.
7. The decimal coding was almost universally accepted as suitable for grasping the essentials of machine language. A few dissidents identified a desire for binary.
8. Less than twenty percent considered the brief study of machine language a waste of time for the ultimate user.
9. All debugging aids provided proved helpful but the greatest aid was discussion with other students.
10. Difficulties with getting ultimately successful runs were most impeded by the actual closed shop mechanism of the Computer Center (probably underqualified monitors, bugs still in the program, and general lack of understanding of procedures). Failure to understand the function of the simulated computer operations and errors in card punching were also substantial contributors.
11. A large majority (over ninety-eight percent) considered the problem set reasonably difficult with the rest equally divided between too hard and too easy.
12. } Problem difficulty was rated roughly equal.
15. }
13. } The most difficult problems took three quarters of the
14. } students less than four hours of homework and less than five computer runs.
16. } The easiest problem took three quarters of the students
17. } less than two hours of homework and less than three computer runs.
18. } Results of this question appear to belie the preceding
19. } two results. For if the program were indeed tested and ready for the run for record it should succeed on the first, or at worst, second run. The statistics indicate that many used four or more of these runs on their more difficult problem.
20. A majority felt that there was a sufficient diversity in the problem set although several felt the problems were too similar.

- 21.} Analysis, coding and debugging difficulty varied much
22.} between individuals and no one stood out as uniformly
particularly hard or particularly easy.

TABLE I

TO: Students who took CS162 during Winter Term 1969

FROM: D. R. Clutterham, Head of Mathematical Sciences Dept.

We need to obtain some information regarding the use of the simulated computer used to teach machine language in the CS162 course. Please complete the following questionnaire as accurately as possible and return to the Mathematical Sciences Department in person or by campus mail. If desired you may delete the portion above the double line to preserve anonymity. Please complete and return immediately.

Underline answer which fits your case.

1. Had you ever worked with machine language before?
(a) yes (b) no
2. The instruction set provided was
(a) too complex, (b) about right, (c) too elementary
3. The arithmetic instructions
(a) were adequate, (b) should have included shifting,
(c) should have included multiplication.
4. The branch instruction set
(a) was adequate, (b) could be improved with some
additional types, (c) had too many alternatives.
5. The input/output set of instructions was
(a) too restrictive, (b) adequate, (c) should permit
formatting
6. Logic instructions should be included
(a) no, (b) such as "AND", "OR", "COMPLEMENT."
7. Greater understanding of machine language would have
been obtained if numbers and codes had been
(a) in octal, (b) in hexadecimal, (c) in binary,
(d) the decimal used was adequate.
8. The study of machine language
(a) is a waste of time for an ultimate user
(b) gave me a much better understanding of computers
(c) contributed to my appreciation of FORTRAN

9. The most helpful debugging aid was
(a) the program trace, (b) the memory and status dump,
(c) discussion with monitor, (d) discussion with
classmates
10. The greatest difficulty in completing a program
successfully was
(a) incomplete understanding of instructions
(b) getting results from a run on the computer
(c) punching an accurate set of cards
11. The problem set to be solved
(a) was adequate, (b) was too difficult,
(c) was too easy
12. The problem which was most difficult for me was
(a) 1 (b) 2 (c) 3 (d) 4 (e) 5
13. The problem which was most difficult for me required
(a) less than 2 hours of homework
(b) two to 4 hours of homework
(c) four to 10 hours of homework
(d) over 10 hours of homework
14. The problem which was most difficult for me required
(a) less than 3 computer runs, (b) 3 to 5 computer runs
(c) 6 to 9 computer runs, (d) more than 9 computer runs
15. The problem which was easiest for me was
(a) 1 (b) 2 (c) 3 (d) 4 (e) 5
(f) don't remember
16. The problem which was easiest for me required
(a) less than 2 hours of homework, (b) 2 to 4 hours
of homework, (c) 4 to 10 hours of homework
(d) over 10 hours of homework
17. The problem which was easiest for me required
(a) less than 3 computer runs, (b) 3 or 4 computer runs
(c) 5 to 7 computer runs, (d) 8 or more computer runs
18. My easiest problem ran correctly on my run for record
number
(a) 1 (b) 2 (c) 3 (d) 4 or greater
19. My hardest problem ran correctly on my run for record
number
(a) 1 (b) 2 (c) 3 (d) 4 or greater
20. The problems in our problem set
(a) were about right, (b) were too similar,
(c) were too different

21. The part of these problems I found easiest was
(a) analysis, (b) coding, (c) debugging
22. The part of these problems I found hardest was
(a) analysis, (b) coding, (c) debugging
23. Include any comments for improving this part of the
course.

Survey of Instructors Using the Automated Problem Sets

Seven instructors have been introduced to the automated problem sets and five have taught the introductory computer course at Florida Institute of Technology using the sets. Their observations are summarized here.

When a class is given a common problem, there is a tendency to either copy the solution of one of the better students or to work collectively on a program so that the net result is several groups of identical solutions. This problem is not peculiar to this course or even this subject, but usually students vary their own solutions from the one they copy and this is not done with the automated problem sets. One solution may be to have the students turn in their handwritten coding sheet before they begin their actual machine debugging; then their final programs should be modifications to the handwritten ones. Another solution is to develop a very large set of similar problems so that students have essentially an individual problem.

The instruction code set seems generally suitable to the instructors. More experienced instructors found the set quite suitable or else desired only a shift operation. Newly indoctrinated instructors desired a multiply and perhaps also a divide instruction. Somewhat more capability in the input-output format appears desirable, although exactly what form it should take was not agreed upon. A set of left and right shifts with and without a circular capability have been designed for the program but are not now incorporated.

One anticipated problem - that of teaching the use of the keypunch in classroom - has not arisen; learning the use of the keypunch seems to be passed very readily between the students, and a minimum of words from the instructor is sufficient.

Survey of Machine Operators Using the Simulator

An initial complaint of the operators was that instructors did not sufficiently define the problem to the students and further definition had to be supplied in detail. This is recognized as a continuing problem and the instructors are putting more care and detail into the definition.

A second difficulty is that assignments are relatively few, but everyone's problem comes due at the same time. Even if the assignments are given well in advance, normal student procrastination causes a heavy run on both the card punching equipment and on the computer in the last couple of days before a grading run is due. A solution to this problem, as yet untried, is to stagger problem due dates giving easier problems to the students whose problems are due first. In addition to this, simpler problems could be given much earlier in the quarter so that the students can first learn some of the mechanics of preparing a problem for the machine and getting basic input-output mastered.

APPENDIX IV

Program Listing


```

// JOB
// *
// *PROGRAM TO READ IN DATA FILE,INITIALIZE AND START SIMULATION
// * OF A DECK OF SIM610 PROGRAMS.
// *
// FOR
*NAME STRTG
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
    INTEGER A(2205),INPUT(160),NREM(77),DATA(212),PRSET(15)
    INTEGER TABLE(16)
    INTEGER ERR,EA
    INTEGER DATA1(106),DATA2(106)
    COMMON A,INPUT,NREM,DATA,PRSET
    EQUIVALENCE (NPROB,A(2140)),(TABLE(1),A(2116))
    EQUIVALENCE (INIT,A(2138))
    EQUIVALENCE (EA,A(2025)),(ERR,A(2109))
    EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
    EQUIVALENCE (LOC11,A(1)),(LOC12,A(1001))
    EQUIVALENCE (NI,A(2114)),(NC,A(2115))
    DEFINE FILE 5(12,106,U,NXRDC)
1  NI=2
    NO=5
    READ(NI,11) TABLE,NDTST,INIT,PRSET
11  FORMAT(16A1,I1,1X,I1,1X,15I2)
    IF(INIT-1) 10,12,10
10  INIT = -1
    GO TO 13
12  INIT = 0
13  IF(NDTST) 16,16,14
14  IF(NDTST-6) 15,15,16
15  READ(5'2*NDTST-1) DATA
    GO TO 19
16  EA=1
    DO 4 I=1,106
        CALL RDR60
        DATA1(I)=LOC11
        DATA2(I)=LOC12
        IF(ERR) 3,2,3
3    PAUSE 7009
        I = I - 1
2    LOC12 = IABS(LOC12)
4    WRITE(NO,17) LOC11,LOC12
17  FORMAT(1H ,I4,I3)
19  CALL RDR60
    CAL. LINK(LOADP)
    END
// DUP
*DELETE          STRTG
*STORECI      WS  UA  STRTG 0001
*FILES(5,SIMDT)

```

```

STRTG001
STRTG002
STRTGC03
STRTGC04
STRTGC05
STRTGC06
STRTGC07
STRTGC08
STRTG009
STRTG010
STRTGC11
STRTG012
STRTG013
STRTG014
STRTG015
STRTG016
STRTG017
STRTG018
STRTGC19
STRTG020
STRTG021
STRTG022
STRTG023
STRTG024
STRTG025
STRTG026
STRTGC27
STRTG028
STRTG029
STRTG030
STRTG031
STRTG032
STRTG033
STRTG034
STRTG035
STRTG036
STRTG037
STRTG038
STRTG039
STRTG040
STRTG041
STRTG042
STRTG043
STRTG044
STRTG045
STRTG046
STRTG047
STRTG048
STRTG049
STRTG050
STRTG051
STRTG052
STRTG053
STRTG054
STRTG055

```

```

// JOB
// *
// * PROGRAM TO LOAD EACH STUDENT PROGRAM INTO PSEUDO CORE, AND
// * BRING IN THE FILE OF STANDARD DATA ON THE PROBLEM FOR GRADING.
// *
// DUP
*DELETE          LOADP
// FOR
*NAME LOADP
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
    INTEGER ERRS
    INTEGER ERROR
    INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMC(2)
    INTEGER IOBUF(48),NAME(32),ERRCT(5)
    INTEGER TABLE(16)
    INTEGER RNTIM(2),PROGL
    INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
    INTEGER NSAV1(30),NSAV2(30)
    INTEGER STORT,STDPL
    INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
    INTEGER NRDSR(10),LOCRO(10)
    INTEGER PTSR,PTSA,PTSW,PTS
    INTEGER FDATA,PCSPT(3)
    INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCO,PTCW(10),PTWO
    INTEGER PCGRT,PCGPL
    INTEGER RDATA(14)
    INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
    COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,NAME
    COMMON ERRCT
    COMMON NI,NO,TABLE,JERR
    COMMON I,J,K,L,M
    COMMON INIT
    COMMON NSTUD,NPROB
    COMMON RNTIM,PROGL,NOCD5
    COMMON NANSW,NSAV1,NSAV2
    COMMON IDUMY,STORT,STDPL,ANS1,ANS2,NORDS,NRGPS,PCSPT
    COMMON NRDSR,LOCRO,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
    COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
    COMMON PTSR,PTSA,PTSW,PTS
    COMMON FILNO,PC,IOVFL,LINE,DATA
    EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
    EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
    EQUIVALENCE (POSPT(1),NPPT), (POSPT(2),NPPTA), (POSPT(3),NPPTW)
    EQUIVALENCE (N1,NOCDS), (NWTR,NANSW)
    EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
    EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
    EQUIVALENCE (ERRCT(1) ERROR)
    DEFINE FILE 1(24,160,U,NXREC)
C-----TEST FOR MONITOR CARD
    1 IF(ERROR) 2,10,10
    2 IF(IOBUF(2)-TABLE(1)) 10,20,10
10 EA=1
    CALL RDR60
    GO TO 1
C-----SKIP TO NEW PAGE, PRINT MONITOR CARD
20 WRITE(NO,22) IOBUF
22 FORMAT(1H1,16A1,32A2,/)
    DO 27 I=2,15
    DO 26 J=1,10

```

```

LOADP001
LOADP002
LOADP003
LOADP004
LOADP005
LOADP006
LOADP007
LOADP008
LOADP009
LOADP010
LOADP011
LOADP012
LOADP013
LOADP014
LOADP015
LOADP016
LOADP017
LOADP018
LOADP019
LOADP020
LOADP021
LOADP022
LOADP023
LOADP024
LOADP025
LOADP026
LOADP027
LOADP028
LOADP029
LOADP030
LOADP031
LOADP032
LOADP033
LOADP034
LOADP035
LOADP036
LOADP037
LOADP038
LOADP039
LOADP040
LOADP041
LOADP042
LOADP043
LOADP044
LOADP045
LOADP046
LOADP047
LOADP048
LOADP049
LOADP050
LOADP051
LOADP052
LOADP053
LOADP054
LOADP055
LOADP056
LOADP057
LOADP058
LOADP059
LOADP060
LOADP061
LOADP062
LOADP063
LOADP064

```

IF(IOBUF(I)-TABLE(J)) 26,27,26	LOADP065
26 CONTINUE	LOADP066
C---ERROR IOBUF(I) SET TO ZERO.	LOADP067
J = 1	LOADP068
27 IOBUF(I) = J-1	LOADP069
C-----TEST FOR MONITOR START CARD	LOADP070
C NCARD = (((IOBUF(2)*10+IOBUF(3))*10+IOBUF(4))*10+IOBUF(5))*10+	LOADP071
C 1IOBUF(6))*10+IOBUF(7)	LOADP072
NCARD = ((IOBUF(4)*10+IOBUF(5))*10+IOBUF(6))*10+IOBUF(7)	LOADP073
C NO LIST OF SOURCE PROGRAM IF NCARD EQUALS ZERO.	LOADP074
IF(NCARD-1) 28,29,10	LOADP075
28 NCARD = 2	LOADP076
29 NOCDS = 1	LOADP077
NSTUD = (((IOBUF(9)*10+IOBUF(10))*10+IOBUF(11))*10+IOBUF(12))*10+I	LOADP078
IOBUF(13)	LOADP079
NPROB = 10*IOBUF(14) + IOBUF(15)	LOADP080
DO 45 I=1,32	LOADP081
45 NAME(I) = IOBUF(I+16)	LOADP082
C	LOADP083
C-----	LOADP084
C ROUTINE TO LOAD STUDENT PROGRAM INTO 1000 WORD PSUEDO-CORE.	LOADP085
C A LISTING IS PRINTED OF ALL NON-MONITOR CARDS.	LOADP086
C MONITOR CARDS ARE IDENTIFIED BY AN ASTERISK IN COLUMN 1.	LOADP087
C ROUTINE RETURNS ON READING A MONITOR CARD, OR WHEN CORE LOAD	LOADP088
C EXCEEDS PSUEDO-CORE.	LOADP089
C ON RETURN--IAR CONTAINS COUNT OF CORE LOCATIONS USED.	LOADP090
C IOBUF CONTAINS LAST RECORD READ.	LOADP091
C ERRS CONTAINS COUNT OF ERROR FLAGS.	LOADP092
C ERRORS ARE FLAGGED WITH AN ASTERISK ON LISTING.	LOADP093
C ERRORS ARE ALWAYS LISTED.	LOADP094
C LOADING STARTS IN CORE LOCATION ZERO.	LOADP095
C	LOADP096
C CLEAR PSUEDO-CORE.	LOADP097
C AREG(1)=20000	LOADP098
C AREG(2)=20000	LOADP099
C DO 3 IAR=1,18	LOADP100
3 XR(IAR) = 25000	LOADP101
DO 106 IAR=1,2000	LOADP102
106 LOC(IAR)=30000	LOADP103
C INITIALIZE IAR AND ERRS.	LOADP104
C IAR=0	LOADP105
C ERRS=0	LOADP106
110 EA = IAR + 1	LOADP107
CALL RDR60	LOADP108
IF(ERROR) 120,140,130	LOADP109
130 ERRS=ERRS+1	LOADP110
C PUT ASTERISK IN ERROR FLAG.	LOADP111
C ERROR=TABLE(15)	LOADP112
C GO TO 150	LOADP113
C BLANK OUT ERROR FLAG	LOADP114
140 ERROR=TABLE(12)	LOADP115
C NO LIST OF SOURCE PROGRAM IF NCARD EQUALS TWO	LOADP116
C GO TO(150,160),NCARD	LOADP117
150 WRITE(NO,51) ERROR,IAR,IOBUF	LOADP118
51 FORMAT(1H ,A1,1X,I4,4X,7A1,4X,9A1,32A2)	LOADP119
160 IAR=IAR+1	LOADP120
C TEST FOR END OF PSUEDO-CORE.	LOADP121
IF(IAR-999) 110,110,120	LOADP122
C-----* CARD OR END OF CORE ENCOUNTERED	LOADP123
120 WRITE(NO,51) TABLE(12),IAR,IOBUF	LOADP124
PROGL=IAR	LOADP125
ERROR = 0	LOADP126
C	LOADP127
C-----ABORT IF MISPUNCHED CARD IN DECK.	LOADP128
IF(ERRS) 30,30,10	LOADP129
30 IF(IOBUF(3)-TABLE(1)) 31,2,31	LOADP130


```

// JOB
// *
// * PROGRAM TO LOAD EACH STUDENT PROGRAM INTO PSEUDO CORE, AND
// * BRING IN THE FILE OF STANDARD DATA ON THE PROBLEM FOR GRADING.
// *
// DUP
*DELETE          LOADP
// FOR
*NAME LOADP
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
    INTEGER ERRS
    INTEGER ERROR
    INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOC,NEUMG(2)
    INTEGER IOBUF(48),NAME(32),ERRCT(5)
    INTEGER TABLE(16)
    INTEGER RNTIM(2),PROGL
    INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
    INTEGER NSAV1(30),NSAV2(30)
    INTEGER STORT,STDPL
    INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
    INTEGER NRDSR(10),LOCRD(10)
    INTEGER PTSR,PTSA,PTSW,PTS
    INTEGER FDATA,POSPT(3)
    INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCO,PTCW(10),PTWO
    INTEGER PCGRT,PCGPL
    INTEGER RDATA(14)
    INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
    COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOC,NEUMO,IOBUF,NAME
    COMMON ERRCT
    COMMON NI,NO,TABLE,JERR
    COMMON I,J,K,L,M
    COMMON INIT
    COMMON NSTUD,NPROB
    COMMON RNTIM,PROGL,NOCDS
    COMMON NANSW,NSAV1,NSAV2
    COMMON IDUMY,STORT,STDPL,ANS1,ANS2,NORDS,NRGPS,POSPT
    COMMON NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
    COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
    COMMON PTSR,PTSA,PTSW,PTS
    COMMON FILNO,PC,IOVFL,LINE,DATA
    EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
    EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
    EQUIVALENCE (POSPT(1),NPPTR),(POSPT(2),NPPTA),(POSPT(3),NPPTW)
    EQUIVALENCE (N1,NOCDS),(NWTR,NANSW)
    EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
    EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
    EQUIVALENCE (ERRCT(1),ERROR)
    DEFINE FILE 1(24,160,U,NXREC)
C-----TEST FOR MONITOR CARD
    1 IF(ERROR) 2,10,10
    2 IF(IOBUF(2)-TABLE(1)) 10,20,10
    10 EA=1
        CALL RDR60
        GO TO 1
C-----SKIP TO NEW PAGE, PRINT MONITOR CARD
    20 WRITE(NO,22) IOBUF
    22 FORMAT(1H1,16A1,32A2,/)
        DO 27 I=2,15
        DO 26 J=1,10

```

```

LOADP001
LOADP002
LOADP003
LOADP004
LOADP005
LOADP006
LOADP007
LOADP008
LOADP009
LOADP010
LOADP011
LOADP012
LOADP013
LOADP014
LOADP015
LOADP016
LOADP017
LOADP018
LOADP019
LOADP020
LOADP021
LOADP022
LOADP023
LOADP024
LOADP025
LOADP026
LOADP027
LOADP028
LOADP029
LOADP030
LOADP031
LOADP032
LOADP033
LOADP034
LOADP035
LOADP036
LOADP037
LOADP038
LOADP039
LOADP040
LOADP041
LOADP042
LOADP043
LOADP044
LOADP045
LOADP046
LOADP047
LOADP048
LOADP049
LOADP050
LOADP051
LOADP052
LOADP053
LOADP054
LOADP055
LOADP056
LOADP057
LOADP058
LOADP059
LOADP060
LOADP061
LOADP062
LOADP063
LOADP064

```


IF(IOBUF(I)-TABLE(J)) 26,27,26	LOADP065
26 CONTINUE	LOADP066
C---ERROR IOBUF(I) SET TO ZERO.	LOADP067
J = 1	LOADP068
27 IOBUF(I) = J-1	LOADP069
C-----TEST FOR MONITOR START CARD	LOADP070
C NCARD = (((IOBUF(2)*10+IOBUF(3))*10+IOBUF(4))*10+IOBUF(5))*10+	LOADP071
C 1IOBUF(6))*10+IOBUF(7)	LOADP072
C NCARD = ((IOBUF(4)*10+IOBUF(5))*10+IOBUF(6))*10+IOBUF(7)	LOADP073
C NO LIST OF SOURCE PROGRAM IF NCARD EQUALS ZERO.	LOADP074
IF(NCARD-1) 28,29,10	LOADP075
28 NCARD = 2	LOADP076
29 NOCDS = 1	LOADP077
NSTUD = (((IOBUF(9)*10+IOBUF(10))*10+IOBUF(11))*10+IOBUF(12))*10+IOBUF(13)	LOADP078
NPROB = 10*IOBUF(14) + IOBUF(15)	LOADP079
DO 45 I=1,32	LOADP080
45 NAME(I) = IOBUF(I+16)	LOADP081
C	LOADP082
C	LOADP083
C-----ROUTINE TO LOAD STUDENT PROGRAM INTO 1000 WORD PSUEDO-CORE.	LOADP084
C A LISTING IS PRINTED OF ALL NON-MONITOR CARDS.	LOADP085
C MONITOR CARDS ARE IDENTIFIED BY AN ASTERISK IN COLUMN 1.	LOADP086
C ROUTINE RETURNS ON READING A MONITOR CARD, OR WHEN CORE LOAD	LOADP087
C EXCEEDS PSUEDO-CORE.	LOADP088
C ON RETURN--IAR CONTAINS COUNT OF CORE LOCATIONS USED.	LOADP089
C IOBUF CONTAINS LAST RECORD READ.	LOADP090
C ERRS CONTAINS COUNT OF ERROR FLAGS.	LOADP091
C ERRORS ARE FLAGGED WITH AN ASTERISK ON LISTING.	LOADP092
C ERRORS ARE ALWAYS LISTED.	LOADP093
C LOADING STARTS IN CORE LOCATION ZERO.	LOADP094
C	LOADP095
C	LOADP096
C CLEAR PSUEDO-CORE.	LOADP097
AREG(1)=20000	LOADP098
AREG(2)=20000	LOADP099
DO 3 IAR=1,18	LOADP100
3 XR(IAR) = 25000	LOADP101
DO 106 IAR=1,2000	LOADP102
106 LOC(IAR)=30000	LOADP103
C INITIALIZE IAR AND ERRS.	LOADP104
IAR=0	LOADP105
ERRS=0	LOADP106
110 EA = IAR + 1	LOADP107
CALL RDR60	LOADP108
IF(ERROR) 120,140,130	LOADP109
130 ERRS=ERRS+1	LOADP110
C PUT ASTERISK IN ERROR FLAG.	LOADP111
ERROR=TABLE(15)	LOADP112
GO TO 150	LOADP113
C BLANK OUT ERROR FLAG	LOADP114
140 ERROR=TABLE(12)	LOADP115
C NO LIST OF SOURCE PROGRAM IF NCARD EQUALS TWO	LOADP116
GO TO(150,160),NCARD	LOADP117
150 WRITE(NO,51) ERROR,IAR,IOBUF	LOADP118
51 FORMAT(1H ,A1,1X,I4,4X,7A1,4X,9A1,32A2)	LOADP119
160 IAR=IAR+1	LOADP120
C TEST FOR END OF PSUEDO-CORE.	LOADP121
IF(IAR-999) 110,110,120	LOADP122
C-----* CARD OR END OF CORE ENCOUNTERED	LOADP123
120 WRITE(NO,51) TABLE(12),IAR,IOBUF	LOADP124
PROGL=IAR	LOADP125
ERROR = 0	LOADP126
C	LOADP127
C-----ABORT IF MISPUNCHED CARD IN DECK.	LOADP128
IF(ERRS) 30,30,10	LOADP129
30 IF(IOBUF(3)-TABLE(1)) 31,2,31	LOADP130

C	SKIP TO NEW PAGE IF LISTING MADE.	LCACP131
	31 GO TO(32,35),NCARD	LCACP132
	32 WRITE(NC,33)	LCACP133
	33 FORMAT(1H1)	LCACP134
C	READ(1'FILNO) IDUMY,STORT,STDPL,ANS1,ANS2,NCRDS,NRGPS,PCSP,PT,	LCACP135
C	1 NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO,	LCACP136
C	2 PTWC,NANS,FDATA,MAXRT,PCGRT,PCGPL(,RDATA)	LCACP137
	35 CALL RDSTD	LCACP138
C	SIMULATE RUN.	LCACP139
	CALL SIMRN	LCACP140
C	DUMP GRADING INFORMATION.	LCACP141
	CALL LINK(DUMPG)	LCACP142
	END	LCACP143
	// DUP	LCACP144
	*STORECI WS UA LOADP 0002	LCACP145
	*LOCAL,RDR6C,CWADD,DECEB	LCACP146
	*FILES(1,FSTDG)	LCACP147

```

// JOB
// *
// *ROUTINE WHICH ACTUALLY SIMULATES EXECUTION OF THE PROGRAM
// * IN PSEUDO-CORE.
// *
// FOR
*LIST ALL
*EXTENDED PRECISION
*ONE WORD INTEGERS
SUBROUTINE SIMRN
  INTEGER TCNTR
  INTEGER SHFTC,CARRY,CARY2
  INTEGER OPTBL(44),NUTBL(44)
  INTEGER AREG1,AREG2
  INTEGER MREG(2)
  INTEGER CXR(2),CXR1,CXR2
  INTEGER          NXREG(2),CEAR(2) ,          NNREG(2)
  INTEGER  IIBUF(7),JJBUFF(7),KKBUF(7),LLBUF(7),MMBUF(7),NNBUF(7)
  INTEGER CEAR1,CEAR2
  INTEGER LOC(2000),XR(18),AREG(2),TAG,ACCR,EA,OPCOC,NEUMC(2)
  INTEGER IOBUF(80),ERRCT(5)
  INTEGER TABLE(16)
  INTEGER RNTIM(2),PROGL
  INTEGER LCC1(1000),LCC2(1000),XR1(9),XR2(9)
  INTEGER NSAV1(30),NSAV2(30)
  INTEGER STDRT,STDPL
  INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
  INTEGER NRDSR(10),LCCRD(10)
  INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWO
  INTEGER FDATA,PCSPT(3)
  INTEGER PCGRT,PCGPL
  INTEGER RDATA(14)
  INTEGER PTSR,PTSA,PTSW,PTS
  INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
  COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ACCR,EA,OPCOC,NEUMC,IOBUF,ERRCT
  COMMON NI,NO,TABLE,JERR
  COMMON I,J,K,L,M
  COMMON INIT
  COMMON NSTUD,NPROB
  COMMON RNTIM,PROGL,NOCD5
  COMMON NANSW,NSAV1,NSAV2
  COMMON IDUMY,STDRT,STDPL,ANS1,ANS2,NCRDS,NRGPS,PCSPT
  COMMON NRDSR,LCCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
  COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
  COMMON PTSR,PTSA,PTSW,PTS
  COMMON FILNO,PC,IOVFL,LINE,DATA
  EQUIVALENCE (LOC(1),LCC1(1)),(LCC(1001),LCC2(1))
  EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
  EQUIVALENCE (POSPT(1),NPPTW),(PCSPT(2),NPPTA),(PCSPT(3),NPPTW)
  PUT INTEGERS USED ONLY HERE IN LINE TO SAVE CORE.
  EQUIVALENCE (LINE(1),IIBUF(1)),(LINE(8),JJBUFF(1)),(LINE(15),
  CKKBUF(1)),(LINE(22),LLBUF(1)),(LINE(29),MMBUF(1)),(LINE(36),
  CNNBUF(1)),(NXREG(1),NXRG1),(NXREG(2),NXRG2)
  EQUIVALENCE (LINE(43),CXR(1)),(LINE(45),MREG(1))
  1,(LINE(47),NXREG(1)),(LINE(49),CEAR(1)),(LINE(51),NNREG(1))
  EQUIVALENCE (LINE(53),TCNTR),(LINE(54),SHFTC)
  EQUIVALENCE (LINE(55),CARRY),(LINE(56),CARY2)
  EQUIVALENCE (LINE(57),MSW ),(LINE(58),LCTR2)
  EQUIVALENCE (LINE(59),IAR ),(LINE(60),IFLAG)
  EQUIVALENCE (CEAR(1),CEAR1),(CEAR(2),CEAR2)
  EQUIVALENCE (CXR(1),CXR1),(CXR(2),CXR2)
  EQUIVALENCE (MREG1,MREG(1)),(MREG2,MREG(2))
  EQUIVALENCE (AREG(1),AREG1),(AREG(2),AREG2)
  EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))

```

```

SIMRNC01
SIMRNC02
SIMRNC03
SIMRNC04
SIMRNC05
SIMRNC06
SIMRNC07
SIMRNC08
SIMRNC09
SIMRNC10
SIMRNC11
SIMRNC12
SIMRNC13
SIMRNC14
SIMRNC15
SIMRNC16
SIMRNC17
SIMRNC18
SIMRNC19
SIMRNC20
SIMRNC21
SIMRNC22
SIMRNC23
SIMRNC24
SIMRNC25
SIMRNC26
SIMRNC27
SIMRNC28
SIMRNC29
SIMRNC30
SIMRNC31
SIMRNC32
SIMRNC33
SIMRNC34
SIMRNC35
SIMRNC36
SIMRNC37
SIMRNC38
SIMRNC39
SIMRNC40
SIMRNC41
SIMRNC42
SIMRNC43
SIMRNC44
SIMRNC45
SIMRNC46
SIMRNC47
SIMRNC48
SIMRNC49
SIMRNC50
SIMRNC51
SIMRNC52
SIMRNC53
SIMRNC54
SIMRNC55
SIMRNC56
SIMRNC57
SIMRNC58
SIMRNC59
SIMRNC60
SIMRNC61
SIMRNC62
SIMRNC63
SIMRNC64

```


EA=ACCR + 1	SIMRN131
C-----SEARCH CP-CODE TABLE FOR MATCH	SIMRN132
IF(CPCOD=10C) 7,30,30	SIMRN133
7 INSTR = C	SIMRN134
8 INSTR = INSTR + 2	SIMRN135
IF(OPCOD=OPTBL(INSTR)) 30,35,8	SIMRN136
C-----INVALID CP-CODE	SIMRN137
30 INSTR = 20	SIMRN138
GO TO 160	SIMRN139
C	SIMRN140
C-----SET CONDITION FLAG	SIMRN141
35 IFLAG = OPTBL(INSTR-1)	SIMRN142
C-----MAKE INSTR EQUAL TO SEQUENCE NO. OF INSTRUCTION.	SIMRN143
INSTR = INSTR/2	SIMRN144
C-----COMPUTE EFFECTIVE ADDRESS	SIMRN145
CXR1 = XR1(TAG)	SIMRN146
CXR2 = XR2(TAG)	SIMRN147
IF(IFLAG-1) 45,40,30	SIMRN148
C-----REQUIRED TAG MISSING	SIMRN149
40 IF(TAG) 160,160,200	SIMRN150
C-----IS INSTRUCTION (ADDRESS) INDEXED	SIMRN151
45 IF(TAG) 200,200,70	SIMRN152
C-----ANY ERRORS...	SIMRN153
160 JERR = 2	SIMRN154
GO TO 57C	SIMRN155
C	SIMRN156
C-----COMPUTE EFFECTIVE ADDRESS FOR INDEXED INSTRUCTIONS.	SIMRN157
70 EA=EA+ CXR(2)+1000	SIMRN158
EA=EA-(EA/1000)*1000	SIMRN159
C	SIMRN160
C	SIMRN161
C-----SAVE CONTENTS OF EA	SIMRN162
200 CEAR1=LCC1(EA)	SIMRN163
CEAR2=LCC2(EA)	SIMRN164
C-----EXECUTE INSTRUCTION	SIMRN165
GO TO (1100,1110,1200,1210,1300,1300,1320,1320,1401,1411,1421,	SIMRN166
C 1500,1510,1520,1530,1540,1600,1610,1770), INSTR	SIMRN167
C	SIMRN168
C	SIMRN169
C	SIMRN170
C-----LOAD ACCUMULATOR. SET SIGN LATCH.	SIMRN171
C	SIMRN172
1100 AREG1=CEAR1	SIMRN173
AREG2=CEAR2	SIMRN174
1105 CALL LATCH(AREG)	SIMRN175
GO TO 500	SIMRN176
C	SIMRN177
C	SIMRN178
C	SIMRN179
C-----STORE ACCUMULATOR. SET SIGN LATCH.	SIMRN180
1110 LOC1(EA)=AREG1	SIMRN181
LOC2(EA)=AREG2	SIMRN182
GO TO 1105	SIMRN183
C	SIMRN184
C	SIMRN185
C-----ADD TO ACCUMULATOR. SET SIGN AND OVFL LATCHES.	SIMRN186
1200 CALL DWADD(AREG,CEAR,AREG,ICVFL)	SIMRN187
GO TO 500	SIMRN188
C	SIMRN189
C	SIMRN190
C-----SUBTRACT FROM ACCUMULATOR. SET SIGN AND OVFL LATCHES.	SIMRN191
1210 NNREG(1)=-CEAR1	SIMRN192
	SIMRN193
	SIMRN194
	SIMRN195
	SIMRN196

NNREG(2)=-CEAR2	SIMRN197
CALL DWADD(AREG,NNREG,AREG,IOVFL)	SIMRN198
GO TO 500	SIMRN199
C	SIMRN200
C	SIMRN201
C-----SHIFT LEFT ACCUMULATOR.	SIMRN202
C-----SHIFT RIGHT ACCUMULATOR	SIMRN203
C-----	SIMRN204
C-----NEGATIVE SHIFT COUNT GIVES INVALID INSTRUCTION.	SIMRN205
1300 IF(CEAR1) 160,1301,1329	SIMRN206
C-----ZERO SHIFT COUNT SETS SIGN LATCH ONLY.	SIMRN207
1301 IF(CEAR2) 160,1373,1302	SIMRN208
1302 IF(CEAR2-6) 1303,1329,1329	SIMRN209
1303 SHFTC = CEAR2	SIMRN210
GO TO 1340	SIMRN211
C-----	SIMRN212
C	SIMRN213
C-----ROTATE LEFT ACCUMULATOR.	SIMRN214
C-----ROTATE RIGHT ACCUMULATOR.	SIMRN215
C-----1000 MOD 6 EQUALS 4.	SIMRN216
C1320 SHFTC = 1000*(CEAR1 - 6*(CEAR1/6)) + CEAR2	SIMRN217
1320 SHFTC = 4*(CEAR1 - 6*(CEAR1/6)) + CEAR2	SIMRN218
SHFTC = SHFTC - 6*(SHFTC/6)	SIMRN219
IF(INSTR-8) 1340,1330,1340	SIMRN220
1330 SHFTC = 6 - SHFTC	SIMRN221
C	SIMRN222
C-----	SIMRN223
C-----ALL SHIFTS	SIMRN224
1340 K = 0	SIMRN225
C-----TO AVOID FORTRAN DIVISION OF NEGATIVE NUMBERS IN SHIFTS.	SIMRN226
IF(AREG2)1346,1345,1350	SIMRN227
1345 IF(AREG1)1347,1350,1350	SIMRN228
1346 AREG2 = -AREG2	SIMRN229
1347 AREG1 = -AREG1	SIMRN230
C-----SAVE FACT THAT SIGN IS NEGATIVE.	SIMRN231
K = 1	SIMRN232
C	SIMRN233
1350 IF(INSTR-6) 1351,1361,1351	SIMRN234
C	SIMRN235
C	SIMRN236
C-----ROTATE INSTRUCTIONS	SIMRN237
C-----SHIFT LEFT ACCUMULATOR.	SIMRN238
1351 DO 1359 I=1,SHFTC	SIMRN239
CARRY = AREG2/100	SIMRN240
AREG2 = (AREG2-100*CARRY)*10	SIMRN241
CARY2 = AREG1/100	SIMRN242
AREG1 = (AREG1-100*CARY2)*10 + CARRY	SIMRN243
IF(INSTR-7) 1356,1358,1358	SIMRN244
C-----SHIFT LEFT ONLY - SET OVERFLOW IF NONZERO DIGIT SHIFTED OUT.	SIMRN245
1356 IF(CARY2) 1357,1359,1357	SIMRN246
1357 IOVFL = 1	SIMRN247
C-----ROTATE INSTRUCTIONS ONLY	SIMRN248
1358 AREG2 = AREG2 + CARY2	SIMRN249
1359 CONTINUE	SIMRN250
GO TO 1371	SIMRN251
C	SIMRN252
C-----SHIFT RIGHT ACCUMULATOR	SIMRN253
1361 DO 1369 I=1,SHFTC	SIMRN254
CARY2 = AREG1/10	SIMRN255
CARRY = AREG1 - 10*CARY2	SIMRN256
AREG1 = CARY2	SIMRN257
1369 AREG2 = AREG2/10 + 100*CARRY	SIMRN258
C RESTORE SIGN OF ACCUMULATOR. SET SIGN LATCH.	SIMRN259
1371 IF(K) 1373,1373,1372	SIMRN260
1372 AREG2 = -AREG2	SIMRN261
AREG1 = -AREG1	SIMRN262

1373 CALL LATCH(AREG)	SIMRN263
GO TO 500	SIMRN264
C-----SHIFT COUNT GREATER THAN SIX	SIMRN265
1329 AREG1 = 0	SIMRN266
AREG2 = 0	SIMRN267
ISIGN = 0	SIMRN268
GO TO 500	SIMRN269
C	SIMRN270
C-----	SIMRN271
C	SIMRN272
C-----	SIMRN273
C	SIMRN274
C-----LOAD XR. SET SIGN LATCH.	SIMRN275
1401 XR1(TAG)=CEAR1	SIMRN276
XR2(TAG)=CEAR2	SIMRN277
CALL LATCH(CEAR)	SIMRN278
GO TO 500	SIMRN279
C	SIMRN280
C-----	SIMRN281
C	SIMRN282
C-----STORE XR. SET SIGN LATCH.	SIMRN283
1411 LOC1(EA)=CXRI	SIMRN284
LOC2(EA)=CXRI	SIMRN285
CALL LATCH(CXR)	SIMRN286
GO TO 500	SIMRN287
C	SIMRN288
C-----	SIMRN289
C	SIMRN290
C-----ADD TO XR. SET SIGN AND OVFL LATCHES.	SIMRN291
1421 CALL DWADD(CXR,CEAR,NXREG,IOVFL)	SIMRN292
XR1(TAG) = NXRG1	SIMRN293
XR2(TAG) = NXRG2	SIMRN294
GO TO 500	SIMRN295
C	SIMRN296
C-----	SIMRN297
C	SIMRN298
C-----UNCONDITIONAL BRANCH.	SIMRN299
1500 IAR = EA - 1	SIMRN300
GO TO 500	SIMRN301
C	SIMRN302
C-----	SIMRN303
C	SIMRN304
C-----BRANCH ON NEGATIVE.	SIMRN305
1510 IF(ISIGN) 1500,500,500	SIMRN306
C	SIMRN307
C-----	SIMRN308
C-----BRANCH ON ZERO.	SIMRN309
1520 IF(ISIGN) 500,1500,500	SIMRN310
C	SIMRN311
C-----	SIMRN312
C	SIMRN313
C-----BRANCH ON POSITIVE.	SIMRN314
1530 IF(ISIGN) 500,500,1500	SIMRN315
C	SIMRN316
C-----	SIMRN317
C	SIMRN318
C-----BRANCH ON OVERFLOW. RESET OVFL LATCH.	SIMRN319
1540 IF(IOVFL) 500,500,1541	SIMRN320
1541 IOVFL=0	SIMRN321
GO TO 1500	SIMRN322
C	SIMRN323
C-----	SIMRN324
C	SIMRN325
C-----READ FROM INPUT DEVICE INTO (EA).	SIMRN326
1600 NOCDS = NOCDS + 1	SIMRN327
IF(NOCDS-NORDS) 1601,1601,1605	SIMRN328

1601 K = NCCDS + FLATA - 1	SIMRN329
LOC1(EA) = DATA1(K)	SIMRN330
LOC2(EA) = DATA2(K)	SIMRN331
GO TO 500	SIMRN332
1605 CALL RDR60	SIMRN333
IF(LKRRCT(1)) 385,500,385	SIMRN334
385 JERR=4	SIMRN335
GO TO 500	SIMRN336
C	SIMRN337
C-----	SIMRN338
C	SIMRN339
C-----WRITE (EA) ONTC OUTPUT DEVICE.	SIMRN340
1610 CALL DECEB(CEAR,KKBUF)	SIMRN341
WRITE(NC,1615)KKBUF	SIMRN342
1615 FORMAT(1H,7A1)	SIMRN343
NWTR=NWTR+1	SIMRN344
IF(NWTR-30) 1617,1617,500	SIMRN345
1617 NSAV1(NWTR)=CEAR1	SIMRN346
NSAV2(NWTR)=CEAR2	SIMRN347
GO TO 500	SIMRN348
C	SIMRN349
C-----	SIMRN350
C	SIMRN351
C	SIMRN352
C-----STOP.	SIMRN353
177C IAR = PC	SIMRN354
JERR=1	SIMRN355
GO TO 500	SIMRN356
C	SIMRN357
C-----	SIMRN358
C	SIMRN359
C	SIMRN360
C-----TRACE IF SSW 1 ON	SIMRN361
500 TCNTR=TCNTR+1	SIMRN362
IF(TCNTR-25) 510,510,501	SIMRN363
501 CALL DATSW(1,J)	SIMRN364
GO TO (510,520),J	SIMRN365
510 MSW=1	SIMRN366
C	SIMRN367
C	SIMRN368
IF(LCTR) 570,560,570	SIMRN369
560 LCTR=1	SIMRN370
WRITE(NC,561)	SIMRN371
561 FORMAT(' XEQNC ADDR C(ADDR) MNEMONIC C(XR) EA',	SIMRN372
C ' C(EA) C(ACC) C(XR) C(EA)',	SIMRN373
C ' SIGN OVFL',/)	SIMRN374
C-----GET C(ADDR)	SIMRN375
570 CALL DECEB(MREG,IIBUF)	SIMRN376
NEUMC(1) = NUTBL(2*INSTR-1)	SIMRN377
NEUMC(2) = NUTBL(2*INSTR)	SIMRN378
C-----GET C(XR)	SIMRN379
IF(TAG) 580,580,585	SIMRN380
580 DO 582 I=1,7	SIMRN381
JJBUF(I) = TABLE(12)	SIMRN382
MMBUF(I) = TABLE(12)	SIMRN383
582 CONTINUE	SIMRN384
GO TO 590	SIMRN385
585 CALL DECEB(CXR,JJBUF)	SIMRN386
NXRG1 = XR1(TAG)	SIMRN387
NXRG2 = XR2(TAG)	SIMRN388
CALL DECEB(NXREG,MMBUF)	SIMRN389
C-----GET C(EA)	SIMRN390
590 CALL DECEB(CEAR,KKBUF)	SIMRN391
NNREG(1)=LOC1(EA)	SIMRN392
NNREG(2)=LOC2(EA)	SIMRN393
CALL DECEB(NNREG,NNBUF)	SIMRN394
C-----GET C(ACC)	

CALL DECEB(AREG,LLBUF)	SIMRN395
C	SIMRN396
EA=EA-1	SIMRN397
WRITE(NC,596) RNTIM(2),PC,IIBUF,NEUMO,TAG,ACDR,JJBUF,EA,KKBUF,	SIMRN398
C LLBUF,MMBUF,NNBUF,ISIGN,IOVFL	SIMRN399
596 FORMAT(1H ,I5,2X,I4,4X,7A1,4X,2A2,I1,1X,I3,4X,7A1,4X,I4,3X,7A1,5X,	SIMRN400
C 7A1,4X,7A1,5X,7A1,6X,I2,5X,I2)	SIMRN401
C	SIMRN402
GO TO 521	SIMRN403
C	SIMRN404
C	SIMRN405
C-----SKIP LINE WHEN DATSW TURNED OFF	SIMRN406
520 GO TO(512,521),MSW	SIMRN407
512 MSW=2	SIMRN408
WRITE(NC,555)	SIMRN409
555 FORMAT(1H)	SIMRN410
521 IF(JERR) 800,523,800	SIMRN411
C-----FLUSH TC NEXT JOB (SIM610) IF SSW 11 ON	SIMRN412
C (OPERATOR JUDGES TIME EXCESSIVE -IF PRINTING IN LOOP	SIMRN413
C WILL NOT BE STOPPED IN REASONABLE TIME BY CCOUNTER.)	SIMRN414
523 CALL DATSW(11,J)	SIMRN415
GO TO (530,600),J	SIMRN416
C	SIMRN417
C-----BEGIN NEXT MACHINE CYCLE	SIMRN418
600 PC = IAR	SIMRN419
C-----FLUSH TC NEXT PROGRAM (AFTER DUMP) IF RUN TIME EXCESSIVE.	SIMRN420
IF(RNTIM(2)-MAXRT) 1000,1000,530	SIMRN421
530 JERR=3	SIMRN422
800 RETURN	SIMRN423
END	SIMRN424
// DUP	SIMRN425
*DELETE	SIMRN426
*STORE WS UA SIMRN	SIMRN427

```

// JOB
// *
// * PROGRAM TO COMPUTE AND PRINT GRADING INFORMATION AND DUMP CORE.
// *
// FOR
*NAME DUMPG
*IOCS(CARD,1403PRINTER,DISK)
*LIST SYMBOL TABLE
*EXTENDED PRECISION
*ONE WORD INTEGERS
C-----SINCE INTEGER SIZE NOT ADEQUATE,
  REAL RWGTM,RWGPL,PPTT
  INTEGER REG(2),REG1,REG2
  INTEGER KBUFF(7)
  INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMO(2)
  INTEGER IOBUF(48),NAME(32),ERRCT(5)
  INTEGER TABLE(16)
  INTEGER RNTIM(2),PROGL
  INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
  INTEGER NSAV1(30),NSAV2(30)
  INTEGER STDRT,STDPL
  INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
  INTEGER NRDSR(10),LOCRD(10)
  INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWO
  INTEGER PCGRT,PCGPL
  INTEGER FDATA,POSPT(3)
  INTEGER RDATA(14)
  INTEGER PTSR,PTSA,PTSW,PTS
  INTEGER FILNC,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
  INTEGER NAM(31),RAWGR
  COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,NAME
  COMMON ERRCT
  COMMON NI,NC,TABLE,JERR
  COMMON I,J,K,L,M
  COMMON INIT
  COMMON NSTUD,NPROB
  COMMON RNTIM,PROGL,NOCDS
  COMMON NANSW,NSAV1,NSAV2
  COMMON IDUMY,STDRT,STDPL,ANS1,ANS2,NORDS,NRGPS,POSPT
  COMMON NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
  COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
  COMMON PTSR,PTSA,PTSW,PTS
  COMMON FILNC,PC,IOVFL,LINE,CATA
  EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
  EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
  EQUIVALENCE (POSPT(1),NPPTR),(POSPT(2),NPPTA),(POSPT(3),NPPTW)
  EQUIVALENCE (KBUFF(1),LINE(1))
  EQUIVALENCE (REG(1),REG1),(REG(2),REG2)
  EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
  EQUIVALENCE (NAM(1),NAME(1))
  DEFINE FILE 2(800,40,U,NXRCC)

C
C-----EXECUTION COMPLETE
C
C--
C-----PTSR      = PCINTS RECIEVED FOR READING.
      4 PTSR = C
C-----PTSA      = POINTS RECIEVED FOR ANSWERS + ANSWER LOCATIONS.
      PTSA = C
C-----PTSW      = POINTS RECIEVED FOR WRITING ANSWERS.
      PTSW = C
C-----FDATA     = FIRST LOCATION IN 'DATA' FROM WHICH INPUT DATA WAS
C      'READ' (FOLLOWING READS WERE FROM SUCCESSIVE LOCATIONS) .
C      DATA CARD 9, WORD 5.

```

```

DUMPG001
DUMPG002
DUMPG003
DUMPG004
DUMPG005
DUMPG006
DUMPG007
DUMPG008
DUMPG009
DUMPG010
DUMPG011
DUMPG012
DUMPG013
DUMPG014
DUMPG015
DUMPG016
DUMPG017
DUMPG018
DUMPG019
DUMPG020
DUMPG021
DUMPG022
DUMPG023
DUMPG024
DUMPG025
DUMPG026
DUMPG027
DUMPG028
DUMPG029
DUMPG030
DUMPG031
DUMPG032
DUMPG033
DUMPG034
DUMPG035
DUMPG036
DUMPG037
DUMPG038
DUMPG039
DUMPG040
DUMPG041
DUMPG042
DUMPG043
DUMPG044
DUMPG045
DUMPG046
DUMPG047
DUMPG048
DUMPG049
DUMPG050
DUMPG051
DUMPG052
DUMPG053
DUMPG054
DUMPG055
DUMPG056
DUMPG057
DUMPG058
DUMPG059
DUMPG060
DUMPG061
DUMPG062
DUMPG063
DUMPG064

```

<pre> ID=FDATA-1 C-----NRGPS = NO OF GROUPS OF READ AREAS DO 704 I=1,NRGPS C-----LOCRD(I) = FIRST LOCATION OF ITH GROUP TO BE READ INTO. C DATA CARD 3 IAR = LOCRD(I) + 1 C-----NRDSR(I) = NO OF READS REQUIRED IN ITH GROUP. C DATA CARD 2 K=NRDSR(I) DO 704 J=1,K ID=ID+1 IF(LOC1(IAR)-DATA1(ID)) 704,702,704 702 IF(LOC2(IAR)-DATA2(ID)) 704,703,704 C-----PTCR(I) = NO OF POINTS FOR READING EACH CARD IN ITH GROUP C DATA CARD 5 703 PTSR=PTSR+PTCR(I) C-----ITH GROUP CONSISTS OF CONSECUTIVE LOCATIONS. 704 IAR=IAR+1 C-----NORDS = NO OF READS REQUIRED IF(NORDS-NOCDS) 706,705,706 C-----PTCRN = NO OF POINTS FOR CORRECT NO OF READS. C DATA CARD 9, WORD 1 705 PTSR=PTSR + PTCRN 706 CONTINUE IF(PTSR+PTCRN-NPPTR) 710,709,709 709 PTSR = PTSR + PTCRN 710 I = 0 DO 730 K = 1,5 L = 1 C-----NANSR(K) = NO OF ANSWERS IN K'TH ANSWER GROUP. C DATA CARD 4, WORDS 6 TO 10 7101 IF(L-NANSR(K)) 7105,7105,730 C-----LCANS(I) = LOCATIONS IN WHICH ANSWERS ARE TO BE PUT C DATA CARD 4, WORDS 1 TO 5 7105 IAR = LCANS(K) + L IF(LOC1(IAR)-30000) 712,711,712 711 IF(LOC2(IAR)-30000) 712,713,712 C-----PTCA(I) = NO OF POINTS FOR AFFECTING ANSWER LOCATIONS C DATA CARD 6 712 PTSA= PTSA+ PTCA(K) 713 I = I + 1 L = L + 1 IF(I-30) 7135,7135,728 7135 IF(LOC1(IAR)-ANS1(I)) 716,714,716 714 IF(LOC2(IAR)-ANS2(I)) 716,715,716 C-----PTCC(I) = NO OF POINTS FOR CORRECT ANSWERS C DATA CARD 7 715 PTSA= PTSA+ PTCC(K) 716 IF(I-NANSW) 7165,7165,724 7165 IF(NSAV1(I)-ANS1(I)) 719,717,719 717 IF(NSAV2(I)-ANS2(I)) 719,718,719 C-----PTCW(I) = NO OF POINTS FOR PRINTING CORRECT ANS. IN CORR.ORDER C DATA CARD 8 718 PTSW= PTSW+ PTWC(K) C-----NANSW = NO OF ANSWERS WRITTEN 719 DO 722 J= 1,NANSW C-----NSAV1,2(I) = ANSWERS WRITTEN BY PROGRAM (FIRST 10) C-----ANS1,2(I) = CORRECT ANSWERS IF(NSAV1(J)-ANS1(I)) 721,720,721 720 IF(NSAV2(J)-ANS2(I)) 721,723,721 721 IF(J-30) 722,724,724 722 CONTINUE GO TO 724 C-----PTWO = NO OF POINTS FOR PRINTING CORRECT ANS. IN ANY ORDER C DATA CARD 9, WORD 3 723 PTSW= PTSW+ PTWC </pre>	<pre> DUMPG065 DUMPG066 DUMPG067 DUMPG068 DUMPG069 DUMPG070 DUMPG071 DUMPG072 DUMPG073 DUMPG074 DUMPG075 DUMPG076 DUMPG077 DUMPG078 DUMPG079 DUMPG080 DUMPG081 DUMPG082 DUMPG083 DUMPG084 DUMPG085 DUMPG086 DUMPG087 DUMPG088 DUMPG089 DUMPG090 DUMPG091 DUMPG092 DUMPG093 DUMPG094 DUMPG095 DUMPG096 DUMPG097 DUMPG098 DUMPG099 DUMPG100 DUMPG101 DUMPG102 DUMPG103 DUMPG104 DUMPG105 DUMPG106 DUMPG107 DUMPG108 DUMPG109 DUMPG110 DUMPG111 DUMPG112 DUMPG113 DUMPG114 DUMPG115 DUMPG116 DUMPG117 DUMPG118 DUMPG119 DUMPG120 DUMPG121 DUMPG122 DUMPG123 DUMPG124 DUMPG125 DUMPG126 DUMPG127 DUMPG128 DUMPG129 DUMPG130 </pre>
---	--

C-----NANS = NO CF ANSWERS REQUIRED	DUMPG131
C DATA CARD 9, WORD 4.	DUMPG132
724 DO 7265 K1=1,5	DUMPG133
L1 = 1	DUMPG134
7241 IF(L1-NANSR(K1)) 7242,7242,7265	DUMPG135
7242 IAR = LCANS(K1) + L1	DUMPG136
L1 = L1 + 1	DUMPG137
IF(LOC1(IAR)-ANS1(I)) 726,725,726	DUMPG138
725 IF(LOC2(IAR)-ANS2(I)) 726,727,726	DUMPG139
726 GO TO 7241	DUMPG140
7265 CONTINUE	DUMPG141
GO TO 728	DUMPG142
C-----PTCO = NO OF POINTS FOR CORRECT ANS IN CORR LCCS IN ANY ORDER.	DUMPG143
C DATA CARD 9, WORD 2	DUMPG144
727 PTSA= PTSA+ PTCO	DUMPG145
728 GO TO 7101	DUMPG146
730 CONTINUE	DUMPG147
C	DUMPG148
C-----PTS = TOTAL NO. OF POINTS RECIEVED.	DUMPG149
PTS = PTSR + PTSA + PTSW	DUMPG150
C	DUMPG151
C	DUMPG152
C	DUMPG153
C NPPTT = NO. OF POSSIBLE PCINTS - TCTAL.	DUMPG154
NPPTT = NPPTT + NPPTA + NPPTW	DUMPG155
PPTT = NPPTT	DUMPG156
RWGTW = 1.	DUMPG157
RWGPI = 1.	DUMPG158
IF(PTS-NPPTT) 742,750,750	DUMPG159
C-----DO NOT COUNT TIME OR LENGTH BETTER THAN STANDARD IF FULL	DUMPG160
C POINTS WERE NOT EARNED.	DUMPG161
742 IF(RNTIM(2)-STDRT) 744,744,743	DUMPG162
743 RWGTW = RWGTW*STDRT/RNTIM(2)	DUMPG163
744 IF(PROGL-STDPL) 760,760,745	DUMPG164
745 RWGPI = RWGPI*STDPL/PROGL	DUMPG165
GO TO 760	DUMPG166
750 RWGTW = RWGTW*STDRT/RNTIM(2)	DUMPG167
RWGPI = RWGPI*STDPL/PROGL	DUMPG168
760 RAWGR= (100-PCGRT-PCGPL+PCGRT*RWGTW+PCGPL*RWGPI)/PPTT*10.*PTS	DUMPG169
GO TO(780,770,770,780),JERR	DUMPG170
770 RAWGR = 3*RAWGR/4	DUMPG171
780 CONTINUE	DUMPG172
C	DUMPG173
C	DUMPG174
C-----ROUTINE TO DUMP PSUEDO-CORE TO PRINTER.	DUMPG175
C	DUMPG176
C LOC IS 1000 WORD PSUEDO-CORE.	DUMPG177
C DUMP IS TEN 711 INTEGERS PER LINE.	DUMPG178
C ALL OF CORE IS DUMPED.	DUMPG179
C	DUMPG180
WRITE(NG,799) NAME,NPROB	DUMPG181
799 FORMAT(1H0,08X,32A2,12X,'PROBLEM NO.',I4)	DUMPG182
IF(NPROB-4)7995,8205,7995	DUMPG183
8205 NANS=0	DUMPG184
7995 GO TO(801,803,805,807),JERR	DUMPG185
801 WRITE(NG,802)	DUMPG186
802 FORMAT(1H0,'EXECUTION COMPLETE')	DUMPG187
GO TO 820	DUMPG188
803 WRITE(NG,804) PC	DUMPG189
804 FORMAT(1H0,'EXECUTION TERMINATED BY INVALID INSTRUCTION AT ',I3)	DUMPG190
GO TO 820	DUMPG191
805 WRITE(NG,806)	DUMPG192
806 FORMAT(1H0,'EXECUTION TERMINATED DUE TO EXCESSIVE RUN TIME')	DUMPG193
GO TO 820	DUMPG194
807 WRITE(NG,808) PC,EA	DUMPG195
808 FORMAT(1H0,'EXECUTION TERMINATED BY INSTR. AT ',I3,' ATTEMPTING TO	DUMPG196
1 READ 1ST CARD OF NEXT PROG. INTO ',I3)	

810	WRITE(10,811) RNTIM(2),STLRT,PRCGL,STDPL,NCCDS,NCRDS,NANSW,NANS	DUMPG197
811	FORMAT(1H0,10X,'RUNTIME',14X,'LENGTH OF DECK',C8X,'NO OF CARDS '	DUMPG198
	1,'READ',06X,'NO OF ANSWERS WRITTEN'/4(05X,'YCURS',C6X,'STANDARD'),	DUMPG199
	2/,3X,8(16,C6X)/)	DUMPG200
	WRITE(10,815) PTSR,NPPTR,PTSA,NPPTA,PTSW,NPPTW,PTS,NPPTT,RAWGR	DUMPG201
815	FORMAT(1H0,C3X,'POINTS RECEIVED FOR---'/05X,'READING DATA',11X,	DUMPG202
	1 'ANS IN CORN LOCATIONS',C4X,	DUMPG203
	2 'WRITING ANSWERS',C9X,'TCTAL',19X,'RAW',/	DUMPG204
	34(05X,'YOURS',C6X,'STANDARD'),4X,'GRADE',/,3X,9(16,C6X)/)	DUMPG205
	CALL DECEB(AREG,KBUFF)	DUMPG206
	WRITE(10,813) ISIGN,ICVFL,KBUFF	DUMPG207
813	FORMAT(1H , 'SIGN ',12,3X,'CVERFLOW ',12,3X,'ACCUMULATOR',2X,7A1)	DUMPG208
C		DUMPG209
C-----	PRINT INDEX REGISTERS.	DUMPG210
	IAR=C	DUMPG211
	J = 8	DUMPG212
	DO 860 K=1,9	DUMPG213
	IAR = IAR + 1	DUMPG214
	REG1=XR1(IAR)	DUMPG215
	REG2=XR2(IAR)	DUMPG216
C-----	CLEAR UNUSED INDEX REGISTERS	DUMPG217
	IF(REG1-25000) 831,832,831	DUMPG218
832	IF(REG2-25000) 831,833,831	DUMPG219
833	DO 834 L=1,7	DUMPG220
	LINE(J)=TABLE(12)	DUMPG221
834	J=J+1	DUMPG222
	GO TO 860	DUMPG223
831	CALL DECEB(REG,LINE(J))	DUMPG224
	J = J + 7	DUMPG225
860	CONTINUE	DUMPG226
	WRITE(10,843) (LINE(J),J=8,70)	DUMPG227
843	FORMAT(1H ,10X,5HI/RS ,9(3X,7A1),/)	DUMPG228
C-----	DUMP PSEUDO - CORE.	DUMPG229
	IAR=0	DUMPG230
	DO 830 I=1,100	DUMPG231
	J=1	DUMPG232
	M=0	DUMPG233
	DO 880 K=1,10	DUMPG234
	IAR = IAR + 1	DUMPG235
	REG1=LOC1(IAR)	DUMPG236
	REG2=LOC2(IAR)	DUMPG237
	IF(REG1-30000) 851,852,851	DUMPG238
852	IF(REG2-30000) 851,853,851	DUMPG239
853	DO 854 L=1,7	DUMPG240
	LINE(J)=TABLE(12)	DUMPG241
854	J=J+1	DUMPG242
	M=M+1	DUMPG243
	GO TO 880	DUMPG244
851	CALL DECEB(REG,LINE(J))	DUMPG245
	J = J + 7	DUMPG246
880	CONTINUE	DUMPG247
C	DO NOT PRINT LINE IF ALL LOCATIONS IN IT UNAFFECTED BY PROGRAM.	DUMPG248
	IF(M-9) 821,821,830	DUMPG249
821	J=IAR-10	DUMPG250
	WRITE(10,822) J,LINE	DUMPG251
822	FORMAT(1H ,13,2X,10(3X,7A1))	DUMPG252
830	CONTINUE	DUMPG253
C-----	ERROR TRAP	DUMPG254
C		DUMPG255
C-----	IF FINAL GRADE RUN, WRITE GRADE INFO ON FILE.	DUMPG256
C-----	IF INITIALIZATION,GOTO INI2G, IF STUD. PROG. GO TO LOAD NEXT.	DUMPG257
	IF(INIT) 885,881,890	DUMPG258
881	READ(2'1) NFILE	DUMPG259
	NFILE = NFILE + 1	DUMPG260
	WRITE(2'1) NFILE	DUMPG261
	WRITE(2'NFILE) NPROB,NSTUD,JERR,RNTIM(2),PRCGL,PTSR,PTSA,PTSW,NAM.	DUMPG262

```

1RAWGR
885 CALL LINK(LOADP)
890 CALL LINK(INI2G)
END
// DUP
*DELETE          DUMPG
*STORECI        WS UA DUMPG 0001
*FILES(2,SMSTU)

```

```

DUMPG263
DUMPG264
DUMPG265
DUMPG266
DUMPG267
DUMPG268
DUMPG269
DUMPG270

```

// *	INITGC01
// *PROGRAM TO INITIALIZE GRADER.	INITGC02
// *	INITGC03
// FOR	INITGC04
*NAME INITG	INITGC05
*IOCS(CARD,DISK,1403 PRINTER)	INITGC06
*EXTENDED PRECISION	INITGC07
*ONE WORD INTEGERS	INITGC08
*LIST SOURCE PROGRAM	INITGC09
*LIST SUBPROGRAM NAMES	INITGC10
*LIST SYMBOL TABLE	INITGC11
INTEGER A(2205),INPUT(160),CRDIN(78),NREM(77),DATA(212)	INITGC12
INTEGER NRDSR(10),TABLE(16)	INITGC13
INTEGER ERR,EA	INITGC14
INTEGER DATA1(106),DATA2(106)	INITGC15
INTEGER FDATA	INITGC16
COMMON A,INPUT,NREM,DATA	INITGC17
EQUIVALENCE (NPROB,A(2140)),(CRDIN(1),INPUT(69)),(TABLE(1),A(2116))	INITGC18
1), (INIT,A(2138)),(NRDSR(1),CRDIN(1)),(NCRCS,INPUT(64)),(NRGPS,INPUT	INITGC19
2T(65))	INITGC20
EQUIVALENCE (EA,A(2025))	INITGC21
EQUIVALENCE(LCC11,A(1)),(LOC12,A(1001))	INITGC22
EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))	INITGC23
EQUIVALENCE (ERR,A(2109))	INITGC24
EQUIVALENCE (NI,A(2114)),(NC,A(2115))	INITGC25
EQUIVALENCE(FDATA,CRDIN(75))	INITGC26
DEFINE FILE 1(24,160,U,NXREC)	INITGC27
DEFINE FILE 5(12,106,U,NXRDC)	INITGC28
1 INIT = 1	INITGC29
DO 8 I=1,160	INITGC30
8 INPUT(I) = 0	INITGC31
NI=2	INITGC32
NO=5	INITGC33
READ(NI,13) TABLE,NDTST	INITGC34
13 FORMAT(16A1,I1)	INITGC35
CALL DATSW(3,J)	INITGC36
GO TO(500,10),J	INITGC37
500 READ(NI,11) NPROB,NRDSR(1),FDATA	INITGC38
GO TO 600	INITGC39
10 READ(NI,11) NPROB	INITGC40
C READ(NI,11) NRDSR,LCCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,	INITGC41
C 1PTWC,NANS,FDATA	INITGC42
READ(NI,11) CRDIN	INITGC43
600 CONTINUE	INITGC44
11 FORMAT(10(I6,2X))	INITGC45
DO 20 I = 1,10	INITGC46
NRGPS=I-1	INITGC47
K = NRDSR(I)	INITGC48
IF(K) 20,21,20	INITGC49
20 NORDS = NORDS + K	INITGC50
NRGPS = 10	INITGC51
C WRITE(1,NPROB) IDUMY,STDRT,STDPL,ANS1,ANS2,NCRCS,NRGPS,NRDSR,	INITGC52
C 1LCCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,PTWC,NANS,FDATA,PCSPT	INITGC53
21 WRITE(1,NPROB) INPUT	INITGC54
IF(NDTST) 16,16,14	INITGC55
14 IF(NDTST-6) 15,15,16	INITGC56
15 READ(5'2*NDTST-1) DATA	INITGC57
GO TO 19	INITGC58
16 EA=1	INITGC59
DO 4 I=1,106	INITGC60
CALL RDR6C	INITGC61
DATA1(I)=LOC11	INITGC62
DATA2(I)=LOC12	INITGC63
IF(ERR) 3,2,3	INITGC64

```

3 PAUSE 7009
  I = I - 1
2 LOC12 = IABS(LCC12)
4 WRITE(NC,17) LOC11,LOC12
17 FORMAT(1H ,14,I3)
19 CALL RDR60
  CALL LINK(LOADP)
  END
// DUP
*DELETE          INITG
*STORECI      WS UA INITG 0001
*FILES(1,FSTDG),(5,SIMDT)

```

```

INITG065
INITG066
INITG067
INITG068
INITG069
INITG070
INITG071
INITG072
INITG073
INITG074
INITG075
INITG076

```



```

// JOB
// *
// * PROGRAM TO FINISH PROBLEM INITIALIZATION PRCCEDURE.
// *
// FOR
*NAME INI2G
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
  INTEGER ERROR
  INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,CPCCC,NEUMC(2)
  INTEGER IOBUF(48),NAME(32),ERRCT(5)
  INTEGER TABLE(16)
  INTEGER RNTIM(2),PROGL
  INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
  INTEGER NSAV1(30),NSAV2(30)
  INTEGER STDRT,STDPL
  INTEGER NRDSR(10),LCCRD(10)
  INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
  INTEGER RDATA(14)
  INTEGER PTRS,PTSA,PTSW,PTS
  INTEGER FDATA,PCSPT(3)
  INTEGER PCGRT,PCGPL
  INTEGER PTRC(10),PTRCN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWC
  INTEGER FILNC,PC,LINE(70),DATA(212)
  INTEGER INPUT(160), IDUMY(1)
  COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,CPCCC,NEUMC,IOBUF,NAME
  COMMON ERRCT
  COMMON NI,NO,TABLE,JERR
  COMMON I,J,K,L,M
  COMMON INIT
  COMMON NSTUD,NPROB
  COMMON RNTIM,PROGL,NOCD5
  COMMON NANSW,NSAV1,NSAV2
  COMMON IDUMY,STDRT,STDPL,ANS1,ANS2,NCRDS,NRGPS,PCSPT
  COMMON NRDSR,LCCRD,LCANS,NANSR,PTRC,PTCA,PTCC,PTCW,PTRCN,PTCC
  COMMON PTWC,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
  COMMON PTRS,PTSA,PTSW,PTS
  COMMON FILNC,PC,IOVFL,LINE,DATA
  EQUIVALENCE (LOC(1),LCC1(1)),(LCC(1001),LCC2(1))
  EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
  EQUIVALENCE (PCSPT(1),NPPTR),(PCSPT(2),NPPTA),(PCSPT(3),NPPTW)
  EQUIVALENCE (NI,NOCD5),(NWTR,NANSW)
  EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
  EQUIVALENCE (ERRCT(1),ERRCR)
  EQUIVALENCE (INPUT(1),IDUMY(1))
  DEFINE FILE 1(24,160,U,NXREC)
  1 IF(INIT-2) 2,101,101
C----- PUT RESULTS OF RUN OF STANDARD INTO 'STANDARD' VARIABLES.
  2 INIT = 2
    I = 0
    DO 730 K =1,5
      J = 1
710 IF(J-NANSR(K)) 720,720,730
720 IAR = LCANS(K) + J
    I = I + 1
    IF(I-30) 725,725,730
725 ANS1(I) = LOC1(IAR)
    ANS2(I) = LCC2(IAR)
    J = J + 1
    GO TO 710

```

```

INI2GC01
INI2GC02
INI2GC03
INI2GC04
INI2GC05
INI2GC06
INI2GC07
INI2GC08
INI2GC09
INI2GC10
INI2GC11
INI2GC12
INI2GC13
INI2GC14
INI2GC15
INI2GC16
INI2GC17
INI2GC18
INI2GC19
INI2GC20
INI2GC21
INI2GC22
INI2GC23
INI2GC24
INI2GC25
INI2GC26
INI2GC27
INI2GC28
INI2GC29
INI2GC30
INI2GC31
INI2GC32
INI2GC33
INI2GC34
INI2GC35
INI2GC36
INI2GC37
INI2GC38
INI2GC39
INI2GC40
INI2GC41
INI2GC42
INI2GC43
INI2GC44
INI2GC45
INI2GC46
INI2GC47
INI2GC48
INI2GC49
INI2GC50
INI2GC51
INI2GC52
INI2GC53
INI2GC54
INI2GC55
INI2GC56
INI2GC57
INI2GC58
INI2GC59
INI2GC60
INI2GC61
INI2GC62
INI2GC63
INI2GC64

```

730 CONTINUE	INI2GC65
NANS = NANSW	INI2GC66
IF (NANS) 770,770,740	INI2GC67
740 DO 760 I=1,NANS	INI2GC68
ANS1(I) = NSAV1(I)	INI2GC69
ANS2(I) = NSAV2(I)	INI2GC70
IF(I-30) 760,770,770	INI2GC71
760 CONTINUE	INI2GC72
770 STDT = RNTIM(2)	INI2GC73
STDPL = PROGL	INI2GC74
CALL LINK(DUMPG)	INI2GC75
C-----PUT RESULTS OF SECOND PASS THRU DUMPG INTO STD VARIABLES	INI2GC76
C AND PUT STANDARD DATA ON FILE.	INI2GC77
101 NPPTR = PTR	INI2GC78
NPPTA = PTSA	INI2GC79
NPPTW = PTSW	INI2GC80
C WRITE(1,NPROB) IDUMY,STDT,STDPL,ANS1,ANS2,NCRDS,NRGPS,NRDSR,	INI2GC81
C 1LOCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,PTWC,NANS,FDATA,PCSPT	INI2GC82
WRITE(1,NPROB) INPUT	INI2GC83
PAUSE 3333	INI2GC84
C RETURN TO INITIALIZE ANOTHER PROBLEM IF SENSE SWITCH 2 ON.	INI2GC85
CALL DATSW(2,J)	INI2GC86
GO TO(200,777),J	INI2GC87
200 CALL LINK(INITG)	INI2GC88
777 STOP 7777	INI2GC89
END	INI2GC90
// DUP	INI2G091
*DELETE	INI2G092
*STORECI WS UA INI2G 0001	INI2G093
*FILES(1,FSTDG)	INI2G094
	INI2G095

```

// JCB
// FOR
*LIST ALL
*ONE WORD INTEGERS
*EXTENDED PRECISION
      SUBROUTINE ROSTD
      INTEGER A(2205), INPUT(160), NREM(75)
      COMMON A, INPUT, NREM
      EQUIVALENCE(NPRCB, A(2140))
      EQUIVALENCE (MAXRT, INPUT(144))
C
C-----THIS ROUTINE READS THE FILE MADE FROM THE STANDARC FOR THE
C      PROBLEM THE STUDENT IS ATTEMPTING.
C
1      MAXRT = 5000
      IF(NPRCB) 5,5,2
2      IF(NPRCB-24) 10,10,5
5      DO 8 I = 1, 103
8      INPLT(I) = C
      NPRCB = 0
      RETURN
10     READ(1,NPROB) INPUT
      RETURN
      END
// DUP
*DELETE
*STORE      WS UA ROSTD

```

```

ROSTD001
ROSTD002
ROSTD003
ROSTD004
ROSTD005
ROSTD006
ROSTD007
ROSTD008
ROSTD009
ROSTD010
ROSTD011
ROSTD012
ROSTD013
ROSTD014
ROSTD015
ROSTD016
ROSTD017
ROSTD018
ROSTD019
ROSTD020
ROSTD021
ROSTD022
ROSTD023
ROSTD024
ROSTD025
ROSTD026
ROSTD027

```



```

// JCB
// FOR
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST ALL
SUBROUTINE DECEB(REG,VECTR)
  INTEGER REG(2),VECTR(7),DATA,BUF
  INTEGER CORE(20),RCORE(94),TABLE(16)
  COMMON CORE,ISIGN,RCORE,TABLE
C-----
  1 VECTR(1) = TABLE(12)
  DATA = REG(1)
  I = 2
  DO 60 J=1,2
    IF(DATA) 10,25,25
  10 VECTR(1) = TABLE(11)
  DATA = -DATA
  25 L = 100
  DO 50 K=1,3
    BUF = DATA/L
    IF(BUF-9) 30,30,100
  30 VECTR(I) = TABLE(BUF+1)
  DATA = DATA - BUF*L
  L = L/10
  50 I = I + 1
  60 DATA = REG(2)
  RETURN
  100 DO 110 I=2,7
  110 VECTR(I) = TABLE(15)
  RETURN
  END
// DUP
*DELETE
*STORE      WS  UA  DECEB

```

```

DECEB001
DECEB002
DECEB003
DECEB004
DECEB005
DECEB006
DECEB007
DECEB008
DECEB009
DECEB010
DECEB011
DECEB012
DECEB013
DECEB014
DECEB015
DECEB016
DECEB017
DECEB018
DECEB019
DECEB020
DECEB021
DECEB022
DECEB023
DECEB024
DECEB025
DECEB026
DECEB027
DECEB028
DECEB029
DECEB030
DECEB031
DECEB032
DECEB033
DECEB034

```

// JOB		DWADD001
// DUP		DWADD002
*DELETE	DWADD	DWADD003
// FOR		DWADD004
*EXTENDED PRECISION		DWADD005
*ONE WORD INTEGERS		DWADD006
*LIST ALL		DWADD007
SUBROUTINE DWADD(A,B,C,ICVFL)		DWADD008
INTEGER A(2),B(2),C(2),CARRY		DWADD009
INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCCD,NEUMC(2)		DWADD010
INTEGER IOBUF(80),ERRCT(5)		DWADD011
COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,CPCCD,NEUMO,IOBUF,ERRCT		DWADD012
C THIS ROUTINE PERFORMS DOUBLE-WORD DECIMAL ADDITION		DWADD013
C SUCH THAT C = A + B		DWADD014
1 IOVFL = 0		DWADD015
10 C(2) = A(2) + B(2)		DWADD016
CARRY = C(2)/1000		DWADD017
C(2) = C(2) - CARRY*1000		DWADD018
C(1) = A(1) + B(1) + CARRY		DWADD019
CARRY = C(1)/1000		DWADD020
C(1) = C(1) - CARRY*1000		DWADD021
IF(CARRY) 25,30,25		DWADD022
25 IOVFL = 1		DWADD023
ISIGN = CARRY		DWADD024
RETURN		DWADD025
C IF NO CARRY CHECK WHETHER SIGNS OF UPPER + LOWER HALF DISAGREE.		DWADD026
C (IF CARRY A + B MUST HAVE HAD SAME SIGN.)		DWADD027
30 M = 1		DWADD028
I = (C(1)/IABS(C(1)))*(C(2)/IABS(C(2)))		DWADD029
IF(I) 32,40,40		DWADD030
32 IF(C(1)) 33,40,35		DWADD031
33 M = -1		DWADD032
35 C(1) = C(1) - M		DWADD033
C(2) = C(2) + M*1000		DWADD034
40 CALL LATCH(C)		DWADD035
RETURN		DWADD036
END		DWADD037
// DUP		DWADD038
*STORE	WS UA DWADD	DWADD039

```

// JOB      0015
// FOR
*LIST ALL
*EXTENDED PRECISION
*ONE WORD INTEGERS
      SUBROUTINE LATCH(REG)
        INTEGER REG(2)
        INTEGER CORE(2020)
        COMMON CORE,ISIGN

```

```

C
C      THIS ROUTINE SETS THE SIGN INDICATOR, ISIGN, TO -1,0,+1
C      ACCORDING TO THE SIGN OF THE DATA IN A.
C

```

```

C      EXAMPLES...
C      REG(1)  REG(2)  ISIGN
C      -999    000     -1    DATA IS NEGATIVE
C      000     -999    -1    DATA IS NEGATIVE
C      000     000     0     DATA IS ZERO
C      000     999     1     DATA IS POSITIVE
C      999     000     1     DATA IS POSITIVE
C

```

```

      IF ( REG(1) ) 30,20,50
20 IF ( REG(2) ) 30,40,50
30 ISIGN=-1
      RETURN
40 ISIGN=0
      RETURN
50 ISIGN=1
      RETURN
      END

```

```

// DUP
*DELETE          LATCH
*STORE           WS UA LATCH

```

```

LATCH001
LATCH002
LATCH003
LATCH004
LATCH005
LATCH006
LATCH007
LATCH008
LATCH009
LATCH010
LATCH011
LATCH012
LATCH013
LATCH014
LATCH015
LATCH016
LATCH017
LATCH018
LATCH019
LATCH020
LATCH021
LATCH022
LATCH023
LATCH024
LATCH025
LATCH026
LATCH027
LATCH028
LATCH029
LATCH030
LATCH031
LATCH032
LATCH033
LATCH034

```

// JOB	INTFGC01
// *	INTFGC02
// * PROGRAM TO INITIALIZE STUDENT GRADE FILE AND CLEAR STANDARD FILE.	INTFGC03
// *	INTFGC04
// DUP	INTFGC05
*DELETE	INTFGC06
// FOR	INTFGC07
*NAME INTFG	INTFGC08
*ONE WORD INTEGERS	INTFGC09
*EXTENDED PRECISION	INTFGC10
*LIST ALL	INTFGC11
*IOCS(DISK)	INTFGC12
INTEGER ONE(160),TWC(40)	INTFGC13
DEFINE FILE 1(24,160,U,NXREC)	INTFGC14
DEFINE FILE 2(800,40,U,NXRCC)	INTFGC15
1 DO 10 I=1,160	INTFGC16
10 ONE(I) = 0	INTFGC17
DO 20 I=1,40	INTFGC18
20 TWO(I) = 0	INTFGC19
NXREC = 1	INTFGC20
NXRCC = 1	INTFGC21
TWO(I) = 1	INTFGC22
WRITE(2,NXRCC) TWO	INTFGC23
TWO(I) = 0	INTFGC24
DO 30 I=1,24	INTFGC25
30 WRITE(1,I) ONE	INTFGC26
DO 40 I=2,800	INTFGC27
40 WRITE(2,NXRCC) TWC	INTFGC28
CALL EXIT	INTFGC29
END	INTFGC30
// XEQ	INTFGC31
L 01	INTFGC32
*FILES(1,FSTDG),(2,SMSTU)	

// JCB	INDFGC01
// *	INDFGC02
// * PROGRAM TO READ A SET OF DATA FOR THE STUDENT PROGRAMS TO 'READ'	INDFGCC3
// * INTO A FILE (ONE OF 12.)	INDFGC04
// *	INDFGC05
// FOR	INDFGC06
*NAME INDFG	INDFGC07
*IUCS(CARD,DISK,1403 PRINTER)	INDFGC08
*EXTENDED PRECISION	INDFGC09
*ONE WORD INTEGERS	INDFGC10
*LIST SOURCE PROGRAM	INDFGC11
*LIST SUBPROGRAM NAMES	INDFGC12
*LIST SYMBOL TABLE	INDFGC13
INTEGER BUFF(2),OBUFF(7)	INDFGC14
INTEGER A(2165),INPUT(160),NREM(77),DATA(212)	INDFGC15
INTEGER DATA1(106),DATA2(106)	INDFGC16
INTEGER TABLE(16)	INDFGC17
INTEGER ERR,EA	INDFGC18
COMMON A,INPUT,NREM,DATA	INDFGC19
EQUIVALENCE (TABLE,A(2116))	INDFGC20
EQUIVALENCE (LOC11,A(1)),(LOC12,A(1001))	INDFGC21
EQUIVALENCE (NI,A(2114)),(NC,A(2115))	INDFGC22
EQUIVALENCE (ERR,A(2109)),(EA,A(2025))	INDFGC23
EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))	INDFGC24
DEFINE FILE 5(12,106,U,NXRDC)	INDFGC25
1 NI=2	INDFGC26
NO=5	INDFGC27
READ(NI,13) TABLE,NDTST	INDFGC28
13 FORMAT(16A1,11)	INDFGC29
DO 8 I=1,212	INDFGC30
8 DATA (I) = 0	INDFGC31
EA=1	INDFGC32
DO 5 I=1,106	INDFGC33
2 CALL RDR60	INDFGC34
IF(ERR) 3,4,3	INDFGC35
3 PAUSE 7009	INDFGC36
GO TO 2	INDFGC37
4 DATA1(I)=LOC11	INDFGC38
DATA2(I)=LOC12	INDFGC39
BUFF(1) = LOC11	INDFGC40
BUFF(2) = LOC12	INDFGC41
CALL DECEB(BUFF,OBUFF)	INDFGC42
WRITE(NC,11) I,OBUFF	INDFGC43
11 FORMAT(1H ,13,3X,7A1)	INDFGC44
5 CONTINUE	INDFGC45
WRITE(5'2*NDTST-1) DATA	INDFGC46
STOP 7777	INDFGC47
END	INDFGC48
// XEQ L 01	INDFGC49
*FILES(5,SIMDT,0015)	INDFGC50
0123456789- ++ 2	INDFGC51

+000023
-003547
+354501
-000006
+002346
000000
+012345
-001278
+024035
-000023
+850043
+012005
+100000
+233245
-000156
-751000
-000245
+120345
-003486
000000
-001597
+043189
-000005
-100035
+145508
000000
-000135
+000020
-000009
+000045
-000054
+003498
+000009
-120005
-000010
+000005
000000
+000501
+000001
+010101
-100045
-000753
-000005
+000348
+000008
-156247
-036475
-102020
+012045
+000125
-010000
-500134
+000010
+000045
-245365
+360000
+000453
+000125
-000063
+003941
+987654
-853240
+500000
+000283

DATA2C01
DATA2C02
DATA2C03
DATA2C04
DATA2C05
DATA2C06
DATA2C07
DATA2C08
DATA2C09
DATA2C10
DATA2C11
DATA2C12
DATA2C13
DATA2C14
DATA2C15
DATA2C16
DATA2C17
DATA2C18
DATA2C19
DATA2C20
DATA2C21
DATA2C22
DATA2C23
DATA2C24
DATA2C25
DATA2C26
DATA2C27
DATA2C28
DATA2C29
DATA2C30
DATA2C31
DATA2C32
DATA2C33
DATA2C34
DATA2C35
DATA2C36
DATA2C37
DATA2C38
DATA2C39
DATA2C40
DATA2C41
DATA2C42
DATA2C43
DATA2C44
DATA2C45
DATA2C46
DATA2C47
DATA2C48
DATA2C49
DATA2C50
DATA2C51
DATA2C52
DATA2C53
DATA2C54
DATA2C55
DATA2C56
DATA2C57
DATA2C58
DATA2C59
DATA2C60
DATA2C61
DATA2C62
DATA2C63
DATA2C64

-004319
-001800
-004400
-000003
-000051
-000051
+000123
+000045
+000123
-001276
-001357
+000252
-000234
-000005
-000453
+000230
+000015
+000456
+499999
-999910
+888889
-000001
000000
+120450
000000
-112000
+100001
+000008
+102250
-000005
+000300
-000060
000000
-000245
+000035
+000202
+000005
+000023
+000008
+000025
+000010
+000014

DATA2065
DATA2066
DATA2067
DATA2068
DATA2069
DATA2070
DATA2071
DATA2072
DATA2073
DATA2074
DATA2075
DATA2076
DATA2077
DATA2078
DATA2079
DATA2080
DATA2081
DATA2082
DATA2083
DATA2084
DATA2085
DATA2086
DATA2087
DATA2088
DATA2089
DATA2090
DATA2091
DATA2092
DATA2093
DATA2094
DATA2095
DATA2096
DATA2097
DATA2098
DATA2099
DATA2100
DATA2101
DATA2102
DATA2103
DATA2104
DATA2105
DATA2106

PAGE 1

// JOB CC26 0015

0015

LOG DRIVE	CART SPEC	CART AVAIL	PHY DRIVE
000C	0026	0026	0001
0001	0C15	0015	000C

// ASM
*LIST

		*****	AM 0005
		*	AM 0010
		* A M (ASSEMBLER MONITOR)	AM 0015
		*	AM 0020
		*****	AM 0025
C122	01500000	ENT AM AM IS CALL ENTRY POINT	AM 0030
		*****	AM 0035
C000	0C00	BAMS BSS E 0 BEGINING OF AMS	AM 0040
C000	31 04C631A3	IGAR1 DSA DATFT DISK PARAMETERS -- LENGTH	AM 0045
		* OF FILE IN WORDS, SECTOR	AM 0050
		* ADDRESS, AND NO OF SECTORS.	AM 0055
C003	0C40	GTBL BSS 64 GARBAGE TABLE IS 64 WORDS	AM 0060
C043	0C40	ATB BSS 64 ADDRESS TABLE IS 64 WORDS	AM 0065
		*****	AM 0070
C084	0C00	TINST BSS E 0 TABLE OF INSTRUCTIONS	AM 0075
C084	0 1100	DC /1100 00000 WAIT	AM 0080
C085	0 1111	DC /1111 GO TO CCMPUTE GRADE	AM 0085
C086	0 5700	DC /5700 00001 XIO	AM 0090
C087	0 9BDF	DC /9BDF STORE=1. STANDARD ADDRESSING.	AM 0095
C088	0 0C00	DC /0000 00010 SL	AM 0100
C089	0 0C00	DC /0000 GO TO XEQ. HAS NO EA. F=SHCRT	AM 0105
C08A	0 0C00	DC /0000 00011 SR	AM 0110
C08B	0 0C00	DC /0000 GO TO XEQ. HAS NO EA. F=SHCRT	AM 0115
C08C	0 0C00	DC /0000 00100 LDS	AM 0120
C08D	0 0C00	DC /0000 GO TO XEQ. HAS NO EA. F=SHCRT	AM 0125
C08E	0 5700	DC /5700 00101 STS	AM 0130
C08F	0 9BDF	DC /9BDF STORE=1. STANDARD ADDRESSING.	AM 0135
C090	0 2200	DC /2200 00110 WAIT	AM 0140
C091	0 2222	DC /2222 GO TO VALID WAIT ROUTINE	AM 0145
C092	0 1100	DC /1100 00111 WAIT	AM 0150
C093	0 1111	DC /1111 GO TO CCMPUTE GRADE	AM 0155
C094	0 5700	DC /5700 01000 BSI	AM 0160
C095	0 9BDF	DC /9BDF STORE=1. STANDARD ADDRESSING.	AM 0165
C096	0 0C00	DC /0000 01001 BSC	AM 0170
C097	0 0C00	DC /0000 GO TO XEQ.	AM 0175
C098	0 1100	DC /1100 01010 WAIT	AM 0180
C099	0 1111	DC /1111 GO TO CCMPUTE GRADE	AM 0185
C09A	0 1100	DC /1100 01011 WAIT	AM 0190
C09B	0 1111	DC /1111 GO TO CCMPUTE GRADE	AM 0195
C09C	0 0C00	DC /0000 01100 LDX	AM 0200
C09D	0 0C88	DC /0088 GO TO XEQ. EX- 1A, LONG IS EA.	AM 0205
C09E	0 5500	DC /5500 01101 STX	AM 0210
C09F	0 9900	DC /9900 STORE=1. STANDARD EX- NO XR.	AM 0215
COA0	0 0C00	DC /0000 01110 MDX	AM 0220
COA1	0 3C38	DC /3038 (3=SPECIAL MDXLO)	AM 0225
COA2	0 1100	DC /1100 01111 WAIT	AM 0230
COA3	0 1111	DC /1111 GO TO CCMPUTE GRADE	AM 0235
COA4	0 4600	DC /4600 10000 A	AM 0240
COA5	0 8ACE	DC /8ACE STANDARD ADDRESSING.	AM 0245
COA6	0 4600	DC /4600 10001 AD	AM 0250

PAGE 2

COA7 C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0255
COA8 O 4600	DC /4600	10010 S	AM 0260
COA9 C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0265
COAA O 4600	DC /4600	10011 SC	AM 0270
COAB C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0275
COAC O 4600	DC /4600	10100 M	AM 0280
COAD C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0285
COAE O 4600	DC /4600	10101 C	AM 0290
COAF O 8ACE	CC /8ACE	STANDARD ADDRESSING.	AM 0295
COB0 O 1100	DC /1100	10110 WAIT	AM 0300
COB1 O 1111	DC /1111	GO TO CCMPUTE GRADE	AM 0305
COB2 C 1100	DC /1100	10111 WAIT	AM 0310
COB3 O 1111	DC /1111	GO TO CCMPUTE GRADE	AM 0315
COB4 O 4600	DC /4600	11000 LD	AM 0320
COB5 C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0325
COB6 O 4600	DC /4600	11001 LDC	AM 0330
COB7 C 8ACE	CC /8ACE	STANDARD ADDRESSING.	AM 0335
COB8 C 5700	DC /5700	11010 STC	AM 0340
COB9 O 9BDF	DC /9BDF	STORE=1. STANDARD ADDRESSING.	AM 0345
COBA O 5700	DC /5700	11011 STD	AM 0350
COBB O 9BDF	DC /9BDF	STORE=1. STANDARD ADDRESSING.	AM 0355
COBC O 4600	DC /4600	11100 AND	AM 0360
COBD O 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0365
COBE O 4600	DC /4600	11101 CR	AM 0370
COBF C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0375
COCO O 4600	DC /4600	11110 ECR	AM 0380
COC1 C 8ACE	DC /8ACE	STANDARD ADDRESSING.	AM 0385
COC2 O 1100	DC /1100	11111 WAIT	AM 0390
COC3 O 1111	DC /1111	GO TO CCMPUTE GRADE	AM 0395
	*****		AM 0400
COC4 0000	IOCCB BSS E 0	IOCC(S) TO SENSE DEVICE	AM 0405
COO4 O 0000	DC 0	UNUSED	AM 0410
COO5 C 2700	DC /2700	CPU DISK	AM 0415
COO6 O 0000	DC 0	UNUSED	AM 0420
COO7 O 8F00	DC /8F00	2310 FIRST DRIVE	AM 0425
COO8 O 0C00	DC 0	UNUSED	AM 0430
COO9 O 9700	DC /9700	2310 SECOND DRIVE	AM 0435
COCA O 0000	DC 0	UNUSED	AM 0440
COCB O 9F00	DC /9F00	2310 THIRD DRIVE	AM 0445
COCC C 0C00	DC 0	UNUSED	AM 0450
COCD C A700	DC /A700	2310 FOURTH DRIVE	AM 0455
	*****		AM 0460
COCE O 0C00	LIST DC 0	LIST OF MONITOR ENTRY PCINTS	AM 0465
COCF O 0C28	DC \$PRET	PRE-CP I/O ERROR TRAP	AM 0470
COO0 O 0C81	DC \$PST1	POST-CP I/O ERROR TRAP L 1	AM 0475
COO1 O 0C85	DC \$PST2	POST-CP I/O ERROR TRAP L 2	AM 0480
COO2 C 0C89	DC \$PST3	POST-CP I/O ERROR TRAP L 3	AM 0485
COO3 C 0C8D	DC \$PST4	POST-CP I/O ERROR TRAP L 4	AM 0490
COO4 C 0C91	DC \$STCP	PROGRAM STOP KEY TRAP L 5	AM 0495
	*****		AM 0500
COO5 002C	MBUF DMES 'R '14XS E T 'M O C E S W T C '		AM 0505
COEB 0012	DMES I N T R U N 'R 'E		AM 0510
	*****		AM 0515
COF4 1 03AC	TADDR DC CGA	ADDRESS WITHIN COMMON	AM 0520
COF5 1 03AC	DC CGA	ADDRESS WITHIN CALL TV	AM 0525
COF6 1 03AC	DC CGA	ADDRESS WITHIN FLOATING ACC	AM 0530
COF7 1 0276	DC N414	ADDRESS WITHIN LIBF TV	AM 0535
COF8 1 03AC	DC CGA	ADDRESS WITHIN UNUSED CORE	AM 0540

PAGE 3

C0F9 1 03AC	CC	CGA	ADDRESS WITHIN ILS AREA	AM 0545
C0FA 1 0276	CC	N414	ADDRESS WITHIN SUBROUTINES	AM 0550
C0FB 1 03AC	CC	CGA	ADDRESS WITHIN AMS PROGRAM	AM 0555
C0FC 1 0273	CC	N410	ADDRESS WITHIN MAINLINE	AM 0560
C0FD 1 026D	CC	N402	ADDRESS WITHIN RESIDENT MON.	AM 0565
C0FE 1 03AC	CC	CGA	ADDRESS WITHIN FIRST FOUR WDS	AM 0570
C0FF 1 020C	CC	XEQ	EA WITHIN COMMON	AM 0575
C100 1 0362	CC	N510	EA WITHIN CALL TV	AM 0580
C101 1 020C	CC	XEQ	EA WITHIN FLOATING ACC	AM 0585
C102 1 034D	CC	N507	EA WITHIN LIBF TV	AM 0590
C103 1 0322	CC	N504	EA WITHIN UNUSED CORE	AM 0595
C104 1 031F	CC	N503	EA WITHIN ILS AREA	AM 0600
C105 1 033C	CC	N506	EA WITHIN SUBROUTINES	AM 0605
C106 1 031F	CC	N503	EA WITHIN AMS PROGRAM	AM 0610
C107 1 020C	CC	XEQ	EA WITHIN MAINLINE	AM 0615
C108 1 02EE	CC	N501	EA WITHIN RESIDENT MONITOR	AM 0620
C109 1 02EB	CC	N500	EA WITHIN FIRST FOUR WORDS	AM 0625

C10A 00CC	CPARM BSS E	0	DISK PARAMETERS	AM 0630
C1CA C 0C00	DC	0	READ CFF OF DISK INTO BUFFER	AM 0635
C10B 1 0CC0	DC	IOAR1	*LOCATED AT IOAR1	AM 0640
C10C C 0C79	D121 DC	121	CONSTANT	AM 0645
C10D C 0C00	PRONG DC	--*	PROBLEM NUMBER	AM 0650
C10E 0 0C00	STUNC DC	--*	STUDENT NUMBER	AM 0655
	*		*OUTSIDE RESIDENT MONITOR	AM 0660
C10F C 0C00	STAND DC	--*	STANDARD PROBLEM INDICATOR	AM 0665
C110 1 0C00	ABAMS DC	BAMS	ADDRESS BEGINING AMS PRGG.	AM 0670
C111 C 0CC1	EONE DC	1	CONSTANT	AM 0675
C112 C 0CCA	D10 DC	10	CONSTANT	AM 0680
				AM 0685
				AM 0690
C113 C 0C00	LCCRE DC	--*	LENGTH CF CORE	AM 0695
C114 0G0C	TBS BSS	0	TABLE OF LENGTHS OF CORE	AM 0700
C114 C 0C00	LCOMM DC	--*	LENGTH CF COMMON	AM 0705
C115 C 0C00	LCLTV DC	--*	LENGTH CF CALL TV	AM 0710
C116 C 0CC6	LFAC DC	6	LENGTH CF FAC AND INDICATORS	AM 0715
C117 0 0C00	LLBTV DC	--*	LENGTH CF LIBF TV	AM 0720
C118 0 0C00	LGAR DC	--*	LENGTH OF UNUSED CORE	AM 0725
C119 C 0C00	LILS DC	--*	LENGTH CF ILS AREA	AM 0730
C11A C 0C00	LSUB DC	--*	LENGTH CF SUBROUTINES	AM 0735
C11B C 0406	LAMS DC	EAMS-BAMS	LENGTH OF AMS PROGRAM	AM 0740
C11C C 0C00	LMAIN DC	--*	LENGTH CF MAINLINE	AM 0745
C11D C 0C00	LCRM DC	--*	LENGTH OF RESIDENT MONITOR	AM 0750
C11E 0 0CC4	DC	4	LENGTH CF XR SECTION	AM 0755
				AM 0760
C11F 1 0406	AEAMS DC	EAMS	ADDRESS OF END AMS PROG.	AM 0765
C120 C 0000	SMALL DC	0	SMALLEST ADDRESS OF ILS	AM 0770
C121 C 0000	EILS DC	--*	END OF ILS AREA	AM 0775
				AM 0780
				AM 0785
				AM 0790
				AM 0795
				AM 0800
				AM 0805
				AM 0810
				AM 0815
				AM 0820
				AM 0825
				AM 0830
C122 C 0C00	AM DC	--*	ENTRY PCINT FOR AM	

PAGE 4

0123 01 65800122	LDX	11	AM	LOAD XR1 WITH PL ADDRESS	AM 0835
0125 00 C5800C00	LD	11	0	LOAD FIRST PARAMETER	AM 0840
0127 0 DCE5	STO		PRONO	STORE AS PROBLEM NUMBER	AM 0845
0128 01 4C080130	BSC	L	WROPN,+	GO TO WROPN IF ZERC OR NEG	AM 0850
012A 0 9CE7	S		D10	SUBTRACT TEN	AM 0855
012B 01 4C300130	BSC	L	WROPN,Z-	GO TO WROPN IF POSITIVE	AM 0860
012D 0 C0DF	LD		PRONO	LOAD PROBLEM NUMBER	AM 0865
012E 0 90E2	S		EONE	SUBTRACT ONE	AM 0870
012F 0 4838	BSC		+Z-	SKIP UNCONDITIONAL	AM 0875
0130 0 1C10	SLA		16	CLEAR ACC TO ZERO	AM 0880
0131 01 84000001	A	L	IOAR1+1	ADD SECTOR ADDRESS	AM 0885
0133 0 18DC	RTE		16	PLACE ACC INTO EXT	AM 0890
0134 0 C0D7	LD		D121	LOAD 121	AM 0895
0135 01 DC000000	STD	L	IOAR1	STORE AS WC CT AND SECTOR	AM 0900
	*			*ADDRESS FOR DISK READ	AM 0905
0137 0 C8D2	LDD		DPARM	LOAD DISK PARAMETERS	AM 0910
0138 00 440000F2	BSI	L	DZ000	GO TO DISK ROUTINE TO READ	AM 0915
	*			*121 WORDS INTO IOAR+2	AM 0920
013A 01 44000395	BSI	L	READ	READ FROM BIT SW INTO ACC	AM 0925
013C 00 E5800001	AND	11	1	LOGICAL AND SECOND PARAMETER	AM 0930
013E 0 D0D0	STO		STAND	STORE AS STANDARD INDICATOR	AM 0935
	*			NOTE - IF THE BIT SWITCHES	AM 0940
	*			*WERE ALL UP, AND THE ST.	AM 0945
	*			*NUMBER WAS -1, THEN THIS IS	AM 0950
	*			*A STANDARD PROBLEM .	AM 0955
013F 00 C5800001	LD	11	1	LOAD SECOND PARAMETER	AM 0960
0141 0 D0CC	STO		STUND	STORE AS STUDENT NUMBER	AM 0965
0142 0 7102	MCX	1	2	MODIFY XR1 BY 2	AM 0970
0143 01 6D0001F3	STX	L1	AMSR+1	STORE XR1 AS RETURN ADDRESS	AM 0975
	*****				AM 0980
	*				AM 0985
	*			INITIALIZE LENGTH VECTOR	AM 0990
	*				AM 0995
	*****				AM 1000
	*			THE VECTOR BEGINING AT TBS IS INITIALIZED FOR	AM 1005
	*			THE PARTICULAR CORE LOAD. THIS VECTOR GIVES	AM 1010
	*			THE LENGTHS OF THE DIFFERENT PARTS OF THE CORE	AM 1015
	*			LOAD FOR USE LATER IN THE PROGRAM IN GIVING	AM 1020
	*			THE EFFECT OF MEMCRY PROTECT FOR CERTAIN OF	AM 1025
	*			THESE SECTIONS OF CORE. THESE SECTIONS ARE	AM 1030
	*			COMMON, CALL TV, FAC, LIBF TV, UNUSED CORE,	AM 1035
	*			ILS AREA, SUBROUTINES, AMS PROGRAM, MAINLINE,	AM 1040
	*			RESIDENT MONITOR, AND INDEX REGISTER AREA.	AM 1045
	*****				AM 1050
	*****				AM 1055
	*			INITIALIZE LENGTH OF LCMSK	AM 1060
	*****				AM 1065
	*			A MASK IS PREPARED CONTAINING BITS SET IN EACH	AM 1070
	*			POSITION WHERE BITS CAN OCCUR IN AN ADDRESS	AM 1075
	*			ON THE BASIS OF THE LENGTH OF CORE. THEN	AM 1080
	*			INDEX 3 IS TESTED. IF IT IS NEGATIVE, THERE	AM 1085
	*			IS NO LIBF TV CR FAC (FLCATING ACC).	AM 1090
	*****				AM 1095
0145 00 C400000E	LD	L	SCORE	LOAD LENGTH OF CORE	AM 1100
0147 0 DCCB	STO		LCORE	STORE AS LENGTH OF CORE	AM 1105
0148 0 90C8	S		EONE	SUBTRACT ONE	AM 1110
0149 01 D400024D	STO	L	LCMSK	STORE AS LEN. CORE MASK	AM 1115
014B 00 6680007B	LDX	12	\$WRD1	LOAD LOADING ACCR. CORE LOAD	AM 1120

PAGE 5

C14D C	6A5F	STX 2	WRD1	STORE AS LOADING ADDR.	AM 1125
C14E C	C2C1	LD X2	'CMON	LOAD LENGTH OF COMMON	AM 1130
C14F C	DCC4	STO	L COMM	STORE AS LENGTH OF COMMON	AM 1135
C150 C	C2CA	LD X2	'XR3X	LOAD PROPER VALUE XR3	AM 1140
C151 C1	D4C003C8	STO L	SPXR3+1	STORE AS PROPER VALUE XR3	AM 1145
C153 C1	4C28016A	BSC L	NLBTV,Z+	GO TO NO LIBF TV IF MINUS	AM 1150
*****					AM 1155
* INITIALIZE LENGTH OF LIBF TV FOR XR3 POS. *					AM 1160
*****					AM 1165
* DETERMINE LENGTH OF LIBF TV, CALL TV, AND *					AM 1170
* FAC. DETERMINE BEGINING OF CALL AND LIBF TV. *					AM 1175
*****					AM 1180
C155 C	C0BD	LD	L CORE	LOAD LENGTH OF CORE	AM 1185
C156 C	9C8D	S	L COMM	SUBTRACT LENGTH OF COMMON	AM 1190
C157 C	9208	S X2	'TVWC	LENGTH OF TRANSFER VECTOR	AM 1195
C158 C1	D400036E	STO L	BLBTV	STORE AS BEGINING OF LIBF T V	AM 1200
C15A C	C20A	LD X2	'XR3X	LOAD PROPER VALUE XR3	AM 1205
C15B C	8C50	A	CON	ADD CONST. FOR LIBF END	AM 1210
C15C C1	9400036E	S L	BLBTV	SUBTRACT LCM LIBF ADDRESS	AM 1215
C15E C	DCB8	STO	LLBTV	STORE AS LENGTH LIBF TRAN.VEC	AM 1220
C15F C1	8400036E	A L	BLBTV	ADD BEGINING OF LIBF T V	AM 1225
C161 C	8C49	A	D6	ADD SIX FOR FAC AREA	AM 1230
C162 C1	D400036D	STO L	BCALL	STORE AS BEGINING OF CALL TV	AM 1235
C164 C	CCAE	LD	L CORE	LOAD LENGTH OF CORE	AM 1240
C165 C1	9400036D	S L	BCALL	SUBTRACT BEGINING OF CALL TV	AM 1245
C167 C	9CAC	S	L COMM	SUBTRACT LENGTH OF COMMON	AM 1250
C168 C	DCAC	STO	LCLTV	STORE AS LENGTH OF CALL TV	AM 1255
C169 C	7C0F	MDX	PLBTV	GO TO PLBTV	AM 1260
*****					AM 1265
* INITIALIZE LENGTH OF LIBF TV FOR XR3 NEG. *					AM 1270
*****					AM 1275
* DETERMINE LENGTH OF LIBF TV, CALL TV, AND *					AM 1280
* FAC. DETERMINE BEGINING OF CALL AND LIBF TV. *					AM 1285
*****					AM 1290
C16A C	C2C8	NLBTV LD	X2 'TVWC	LOAD LENGTH OF TRANS VECTOR	AM 1295
C16B C	9C3F	S	D6	SUBTRACT SIX	AM 1300
C16C C	D208	STO X2	'TVWC	STORE AS LENGTH OF TRANS VECT	AM 1305
C16D C	DCA7	STO	LCLTV	STORE AS LENGTH OF CALL TV	AM 1310
C16E C	1C1C	SLA	16	ENTER ACC WITH ZERO	AM 1315
C16F C	DCA7	STO	LLBTV	STORE AS LENGTH OF LIBF TV	AM 1320
C170 C	DCA5	STO	LFAC	STORE AS LENGTH OF FAC	AM 1325
C171 C	CCA1	LD	L CORE	LOAD LENGTH OF CORE	AM 1330
C172 C	92C1	S X2	'CMON	SUBTRACT LENGTH OF COMMON	AM 1335
C173 C1	94000115	S L	LCLTV	SUBTRACT LENGTH OF CALL TV	AM 1340
C175 C1	D4C0036D	STO L	BCALL	STORE AS BEGINING OF CALL TV	AM 1345
C177 C1	D400036E	STO L	BLBTV	STORE AS BEGINING OF LIBF TV	AM 1350
*****					AM 1355
* INITIALIZE ADDRESS OF END OF CORE LOAD *					AM 1360
*****					AM 1365
* DETERMINE END OF CORE LOAD (IE. END OF ILS *					AM 1370
* AREA) AND USE AS INITIAL BEGINING ILS AREA. *					AM 1375
*****					AM 1380
C179 C	CC33	PLBTV LD	WRD1	LOAD LOADING ADDR CORE LCAC	AM 1385
C17A C	82C9	A X2	'WCNT	ADD LENGTH OF CORE LOAD	AM 1390
C17B C	92C8	S X2	'TVWC	SUBTRACT LENGTH OF TRANS VECT	AM 1395
C17C C	DCA3	STO	SMALL	STORE AS INITIAL BILS AREA	AM 1400
C17D C	DCA3	STO	EILS	STORE AS END ILS AREA	AM 1405
C17E C1	D40003A8	STO L	DMP+4	STORE AS PARAMETER FOR CUMP	AM 1410

		*****	AM 1415
		* INITIALIZE LENGTH OF RESIDENT MONITOR *	AM 1420
		*****	AM 1425
		* DETERMINE LENGTH OF RESIDENT MONITOR FROM *	AM 1430
		* BEGINING OF CORE TO END OF CORE IMAGE HEADER. *	AM 1435
		*****	AM 1440
0180 C	CC2C	LD WRD1 LCAC LOADING ADDR. CORE LCAC	AM 1445
0181 C	8204	A X2 HWET ADD LENGTH OF CORE IMM HEADER	AM 1450
0182 C	D09A	STO LCRM STORE AS LENGTH OF RES. MON.	AM 1455
		*****	AM 1460
		* INITIALIZE LENGTH OF MAINLINE *	AM 1465
		*****	AM 1470
		* DETERMINE LENGTH OF MAINLINE FROM END OF CORE *	AM 1475
		* IMAGE HEADER TO BEGINING OF AMS. *	AM 1480
		*****	AM 1485
0183 C	C08C	LD ABAMS LOAD ADDR BEGINING OF AMS	AM 1490
0184 C	9C98	S LCRM SUBTRACT LENGTH CORE RES. MON.	AM 1495
0185 C	D096	STO LMAIN STORE AS LENGTH OF MAINLINE	AM 1500
		*****	AM 1505
		* INITIALIZE LENGTH OF SUBROUTINES *	AM 1510
		*****	AM 1515
		* DETERMINE BEGINING OF ILS AREA FROM VALUES IN *	AM 1520
		* THE INTERRUPT TV THAT DO NOT LIE IN THE *	AM 1525
		* MONITOR. DETERMINE THE LENGTH OF SUBROUTINE *	AM 1530
		* AREA FROM THE END OF AMS TO THE BEGINING OF *	AM 1535
		* THE ILS AREA. *	AM 1540
		*****	AM 1545
0186 C	6105	LDX 1 5 ENTER INDEX 1 WITH 5	AM 1550
0187 C	C107	IN1 LD X1 7 LOAD VALUE FROM INTER. TV	AM 1555
0188 C	9024	S WRD1 SUBTRACT LOADING ADDRESS	AM 1560
0189 C1	4C280191	BSC L XX,Z+ GO TO XX ON MINUS	AM 1565
018B C	C107	LD X1 7 LOAD VALUE FROM INTER. TV	AM 1570
018C C	9C93	S SMALL SUBTRACT SMALLEST FOUND	AM 1575
018D C1	4C100191	BSC L XX,- GO TO XX ON NOT MINUS	AM 1580
018F C	C107	LD X1 7 LOAD VALUE FROM INTER. TV	AM 1585
0190 C	DC8F	STO SMALL STORE INTO SMALL	AM 1590
0191 C	71FF	XX MDX 1 -1 MODIFY XRI BY -1, SKIP IF ZERO	AM 1595
0192 C	7CF4	MDX IN1 GO TO IN1	AM 1600
0193 C	C08C	LD SMALL LCAC ILS AREA ADDRESS	AM 1605
0194 C	908A	S AEAMS SUBTRACT ADDRESS END AMS	AM 1610
0195 C	DC84	STO LSUB STORE AS LENGTH SUBROUTINES	AM 1615
		*****	AM 1620
		* INITIALIZE LENGTH OF ILS AREA *	AM 1625
		*****	AM 1630
		* DETERMINE LENGTH OF ILS AREA FROM PREVIOUSLY *	AM 1635
		* DETERMINED BEGINING OF ILS AREA TO END OF *	AM 1640
		* CORE LOAD. *	AM 1645
		*****	AM 1650
0196 C	CC8A	LD EILS LOAD PRCG END ADDRESS	AM 1655
0197 C	9088	S SMALL SUBTRACT BILS AREA ADDRESS	AM 1660
0198 C	D080	STO LILS STORE AS LENGTH ILS AREA	AM 1665
		*****	AM 1670
		* INITIALIZE LENGTH OF UNUSED CORE (GARBAGE) *	AM 1675
		*****	AM 1680
		* DETERMINE LENGTH OF UNUSED CORE (GARBAGE) *	AM 1685
		* FROM END OF CORE LOAD (IE END OF ILS AREA) TO *	AM 1690
		* BEGINING OF LIBF TV. DECREASE LENGTH OF CORE *	AM 1695
		* RESIDENT MONITOR BY FOUR TO LEAVE SPACE FOR *	AM 1700

PAGE 7

C199 C1 C400036E
C19B C 9C85
C19C C1 C4000118
C19E C1 74FCC11D

C1A0 CC C400000D
C1A2 C DC54
C1A3 C CC2C
C1A4 CC D400000D
C1A6 C C218

C1A7 C DC24
C1A8 C CC22

C1A9 C D218

C1AA C 7C13

C1AB C 0006
C1AC C 007A
C1AD C 0000

C1AE C 1000
C1AF C 0818
C180 C1 74C101C8
C182 C 3000
C183 C 0000

C184 C 0811
C185 C E000
C186 C1 4C200183
C188 C1 74FF01C4
C18A C 7005
C18B C 3000
C18C C C00D
C18D C D00A
C18E C C004
C18F C D004
C1C0 C1 4C4001AE

```

* INDEX REGISTER (XR) AREA. * AM 1705
***** AM 1710
LD L BLBTV LOAD BEGINING CF LIBF TV AM 1715
S EILS SUBTRACT PROG END ADDRESS AM 1720
STC L LGAR STORE AS LENGTH OF GARBAGE AM 1725
MDX L LCRM,-4 SUBTRACT 4 FROM LEN. RES PCN AM 1730
***** AM 1735
* PRINT OPERATOR MESSAGE AND RETURN * AM 1740
***** AM 1745
* PRINT OUT THE OPERATOR MESSAGE ON THE CONSOLE * AM 1750
* PRINTER 'SET MODE SW TO INT RUN'. MESSAGE IS * AM 1755
* NOT PRINTED IF INTERRUPT RUN MODE IS ALREADY * AM 1760
* ON. IF OPERATOR DOES NOT CHANGE MODE TO INT * AM 1765
* RUN BEFORE PRESSING PROG START, MESSAGE IS * AM 1770
* PRINTED AGAIN. PRESSING PROGRAM STOP DOES NOT * AM 1775
* CAUSE THIS FUNCTION TO BE ALTERED. WHEN MODE * AM 1780
* SWITCH IS PROPERLY SET, PROGRAM BEGINS TRACING * AM 1785
* THROUGH THE MAINLINE, WITH A INTERRUPT OCCUR- * AM 1790
* ING ON LEVEL 5 BEFORE EACH INSTRUCTION. * AM 1795
* THE LEVEL 5 INTERRUPT ENTRY PCNT IS ENT5. * AM 1800
***** AM 1805
LD L 13 LOAD FROM LCC 13 (L 5 INT TV) AM 1810
STC SAVL5 SAVE IN SAVL5 AM 1815
LD ANL5 LOAD ADDRESS FOR NEW LEVEL 5 AM 1820
STC L 13 STORE AS LEVEL 5 INTER. ACCR. AM 1825
LD X2 'ITCK LOAD 1130 CONSOLE/KEYBOARD AM 1830
* *ISS TV ENTRY AM 1835
STC SAVKC SAVE IN SAVKC AM 1840
LD ANKC LOAD ADDRESS CF NEW CON/KEYBD AM 1845
* *ISS ROUTINE AM 1850
STC X2 'ITCK STORE AS 1130 CON/KEYBD ISS AM 1855
* *TV ENTRY AM 1860
MDX INITX GO TO INITX AM 1865
***** AM 1870
D6 DC 6 CONSTANT AM 1875
CON DC /80-6 CONSTANT AM 1880
WRD1 DC *-* LOADING ADDRESS CORE LOAD AM 1885
* *(BEGINING CORE IMAGE HEADER) AM 1890
***** AM 1895
PRINT NOP NO-CP INSTRUCTION AM 1900
XIO IOCCP PRINT ONE LETTER AM 1905
MCX L IOCCP,1 INCRIMENT ADDRESS IN IOCCP AM 1910
WAIT WAIT FOR INTERRUPT AM 1915
NKC DC *-* ENTRY PT FOR ISS CON/KEYBD AM 1920
* *(ALSC A WAIT INSTRUCTION) AM 1925
XIO IOCC4 SENSE DSW AND RESET ILSW BIT AM 1930
AND HOCC0 REMCVE ALL BUT BUSY IND. AM 1935
BSC L NKC,Z GO TO NKC IF NOT ZERO AM 1940
MDX L PCNT,-1 MODIFY PCNT BY -1,SKIP IF C AM 1945
MDX BCSCP GO TO BCSCP IF NO SKIP AM 1950
WAIT WAIT FOR OPERATOR AM 1955
LD AMBUF LOAD ADDRESS CF MBUF AM 1960
STC IOCCP STORE INTO IOCCP TO RESTORE AM 1965
INITX LC D31 LOAD 31 AM 1970
STC PCNT STORE AS PRINT COUNT AM 1975
BCSCP BOSC L PRINT GO TO PRINT AND OFF INTERRUPT AM 1980
* *UNLESS LEVEL 5 ALSC ON, AM 1985
* *IN WHICH CASE GO TO NL5. AM 1990

```

01C2 0	0C00	H0C00 DC	/0C00	CONSTANT	AM 1995
01C3 0	001F	D31 DC	31	CONSTANT	AM 2000
01C4 0	0C00	PCNT DC	*--	PRINT CCUNT (NC OF CHAR.)	AM 2005
01C6 0	0000	IOCC4 BSS E	0	IOCC TO SENSE CSW AND	AM 2010
01C6 0	0000	DC	0	*RESET CSW AND ILSW	AM 2015
01C7 0	0F01	DC	/0F01	*FOR CONSOLE PRINTER.	AM 2020
01C8 0	0000	IOCCP BSS E	0	IOCC TO PRINT CN CONSOLE	AM 2025
01C8 1	00D5	DC	MBUF	*PRINTER ONE CHARACTER AT	AM 2030
01C9 0	0900	DC	/0900	*LOCATION MBUF	AM 2035
01CA 1	00D5	AMBUF DC	MBUF	ADDRESS OF MBUF	AM 2040
01CB 1	01B3	ANKC DC	NKC	ADDRESS CF NKC	AM 2045
01CC 0	0C00	SAVKC DC	*--	LOCATION TO SAVE K/C ISS ENT	AM 2050
01CE 0	0C02	ICAR2 BSS E	2	TOP OF DISK BUFFER 2	AM 2055
01D0 1	01D2	ANL5 DC	NL5	ADDRESS OF NEW LEVEL 5	AM 2060
01D1 1	024E	AENT5 DC	ENT5	ADDRESS CF ENT5 ENTRY PT	AM 2065
01D2 0	0C00	NL5 DC	*--	NEW ENTRY PCINT FOR LEVEL 5	AM 2070
01D3 0	0862	XIO	IOCC	SENSE DEVICE STATUS WCRD L 5	AM 2075
01D4 0	1001	SLA	1	SHIFT INT RUN BIT INTO ACC C	AM 2080
01D5 01	4C1001C0	BSC L	BOSCP,-	GO TO BOSCP IF NOT INT RUN	AM 2085
01D7 0	CCF9	LD	AENT5	LOAD ENTRY ADDRESS FOR L 5	AM 2090
01D8 00	D4000C0D	STO L	13	STORE INTO LCC 13	AM 2095
01DA 0	CCF1	LD	SAVKC	LOAD SAVED CON/KEYBD ISS ENT	AM 2100
01DB 0	D218	STO X2	ITCK	RESTORE CON/KEYBD ISS TV ENT	AM 2105
01DC 01	4400037F	BSI L	IONC	WAIT FOR ALL I/C CFF	AM 2110
01DE 01	C4000368	LC L	DM1	LOAD MINUS ONE	AM 2115
01E0 01	94000114	S L	LCCMM	SUBTRACT LENGTH OF COMMON	AM 2120
01E2 0	D0C6	STO	BCOMM+1	STORE AS BEGINING OF COMMON	AM 2125
01E3 01	65800114	LDX I1	LCCMM	LOAD XR1 WITH LEN. CF COMMON	AM 2130
01E5 0	6287	LDX 2	-121	ENTER XR2 WITH -121	AM 2135
01E6 01	C600007B	GAGN LD	L2 GTBL+120	LOAD VALUE FROM BUFFER	AM 2140
01E8 00	D5000000	BCOMM STO	L1 *--	STORE IN COMMON	AM 2145
01EA 0	7201	MDX 2	+1	MODIFY XR2 BY 1, SKIP IF ZERO	AM 2150
01EB 0	7C02	MDX	ARCUN	GO TO ARCUN (IF NO SKIP)	AM 2155
01EC 01	74C201EF	MDX L	INSCH,+2	MODIFY BRANCH ADDRESS BY +2	AM 2160
01EE 0	71FF	AROUN MDX	1 -1	MODIFY XR1 BY -1,SKIP IF ZERO	AM 2165
01EF 0	70F6	INSCH MDX	GAGN	GO TO GAGN (IF NO SKIP)	AM 2170
01F0 0	4878	BCSC	+Z	SKIP AND CFF INTERRUPT	AM 2175
01F1 0	1000	NCP		NO-CP	AM 2180
01F2 00	4C000C00	AMSR BSC L	*--	EXIT FROM AMS AND RETURN	AM 2185
01F4 0	0C10	H0010 DC	/0010	CONSTANT	AM 2190
01F5 0	8000	H8000 DC	/8000	CONSTANT	AM 2195
01F6 0	3000	H3000 DC	/3000	CONSTANT (EQUALS WAIT INSTR)	AM 2200
01F7 0	0000	SAVL5 DC	*--	LOCATION TO SAVE L 5 TV	AM 2205
01F8 0	0000	WAITC DC	0	WAIT IF NEGATIVE	AM 2210
01F9 0	0000	WAITD DC	0	WAIT INDICATOR FOR CON ENT SW	AM 2215
01FA 0	0C00	*		WAIT IF CON ENT SW C IS LP	AM 2220
01FA 1	01F9	BSS E	0	EVEN CORE BOUNDARY	AM 2225
01FB 0	3A00	ICCCD DC	WAITD	READ INTO WAITC	AM 2230
01FC 0	1000	DC	/3A00	THE CONSOLE ENTRY SWITCHES	AM 2235
01FD 0	0C00	NOP NCP		A NO-CP INSTRUCTION	AM 2240
01FE 0	0CC2	LADDR DC	0	ADDRESS CF LAST INSTRUCTION	AM 2245
		LINST BSS E	2	LAST INSTRUCTION	AM 2250


```

*
*   X E Q   GO BACK AND EXECUTE NEXT INSTR.
*
*****
* THIS ROUTINE IS ENTERED WHEN IT IS DECIDED TO
* GO ON AND EXECUTE THE NEXT INSTRUCTION.
*****
* INCREMENT INSTRUCTION COUNTER AND TEST
*****
C200 C 6218 XEQ LDX 2 /1B ENTER INDEX 2 WITH 1B HEX
C201 C1 44C00395 BSI L READ READ CON. ENTRY SW INTO ACC
C203 C ECFC AND H0010 REMOVE ALL BUT BIT ELEVEN
C204 C1 4C2003AC BSC L CGA,Z GO TO CGA IF NOT ZERO
C206 C C831 LDD INSC LOAD INSTRUCTION COUNTER
C207 C 9834 SD RTIME SUBTRACT DOUBLE FROM RTIME
C208 C1 4C1803AC BSC L CGA,-+ GO TO COMPUTE GRADE IF ZERO
C20A C C82D LDD INSC LOAD INSTRUCTION COUNTER
C20B C 882E AD DONE ADD DOUBLE ONE
C20C C D82B STD INSC STORE DOUBLE INTO INST. CTR
C20D C 18C6 RTE 6 MOVE LOW 6 OF EXT TO HIGH ACC
C20E C 18CA SRA 16-6 SHIFT THESE BITS INTO LOW ACC
C20F C0 D4000C01 STO L 1 STORE ACC INTO INDEX 1
C211 C1 C4C002C1 LD L ADDR LOAD ADDRESS OF INSTR
C213 C1 D5C00C43 STO L1 ATB STORE INTO ADDRESS TABLE
C215 C CCE6 LD NOP LOAD A NO-OP INSTRUCTION
C216 C DC11 STO WAIT STORE INTO LOCATION WAIT
C217 C 08E2 XIO IOCCO SENSE CONSOLE SWITCHES
C218 C C81D XIO IOCC SENSE DEVICE STATUS WORD L 5
C219 C E8CF CR WAITC LOGICAL OR IN CON ENT SW.
C21A C E8DD CR WAITC LOGICAL OR IN WAITC INDICATOR
* WAITC IS NEG. IF INSTR=WAIT
C21B C1 4C100227 BSC L NWAIT,- GO TO NWAIT IF NOT MINUS
C21D C1 C40002C1 LD L ADDR LOAD ADDRESS OF INSTR
C21F C 1CC4 SLA 4 REMOVE 4 HIGH BITS, SET CARRY
C220 C 1804 SRA 4 RIGHT JUSTIFY ACC
C221 C 4802 BSC C SKIP IF CARRY OFF
C222 C E8D2 CR H80C0 CR IN HIGH ORDER BIT
C223 C E8D2 OR H30C0 MAKE INTO WAIT INSTRUCTION
C224 C DCC3 STO WAIT STORE AS WAIT INSTRUCTION
C225 C1 440C037F BSI L IONC WAIT FOR ALL I/C OFF
C227 C 4CC3 NWAIT BSI RESTC RESTORE ACC,EXT,XR1,XR2,STATS
C228 C 0CCC WAIT DC *-* EITHER A NOP INSTR, OR A
* *WAIT FOR PROGRAM START KEY
* *AND DISPLAY IAR IN STORAGE
* *BUFFER REGISTER
C229 C1 4CCC024E BOSC I ENT5 RETURN AND CFF INTERRUPT
*****
* RESTO - RESTORES ACC,EXT,XR1,XR2,C,C
*****
* THIS ROUTINE IS ENTERED WHEN IT IS DESIRED TO
* RESTORE THE REGISTERS TO THEIR VALUES WHEN THE
* LAST INSTRUCTION WAS EXECUTED. THE ACCUMUL-
* ATOR, EXTENSION, CARRY, CVERFLW, INDEX1, AND
* INDEX 2 ARE RESTORED. (INDEX 3 DOES NOT NEED
* RESTORING AS IT IS NOT ALTERED.)
*****
C22B C CC0C RESTO CC *-* ENTRY PCINT FOR RESTO

```


PAGE 10

022C CC 65C00C0C	SAVX1 LCX L1 *-*	RESTORE INDEX 1	AM 2575
022E CC 66C00C0C	SAVX2 LCX L2 *-*	RESTORE INDEX 2	AM 258C
0230 C C8C3	LDD SAVE1	RESTORE ACC AND EXT	AM 2585
0231 C 2C0C	SAVCO LDS *-*	RESTORE CARRY AND OVERFLOW	AM 2590
0232 C1 4C80022B	BSC I RESTC	RETURN TO CALLING POINT	AM 2595
	*****		AM 260C
	* CONSTANTS FOR USE BY UPPER HALF *		AM 26C5
	*****		AM 261C
0234 0002	SAVE1 BSS E 2	LOCATION TO SAVE ACC EXT	AM 2615
0236 C 0000	ICCC DC 0	ICCC TO SENSE DEVICE	AM 262C
0237 C 3FC1	DC /3FC1	STATUS WORD FOR STOP/INT RUN	AM 2625
0238 C 0000	INSCT DC 0	INSTRUCTION COUNTER	AM 2630
0239 C 0C0C	CC 0	SECOND HALF INSTR. COUNTER	AM 2635
023A C 00C0	DONE CC 0	FIRST WORD OF DOUBLE PRE. 1	AM 2640
023B C 0C01	ONE DC 1	CONSTANT	AM 2645
023C C 00C1	RTIME DC 1	GIVE ONE MINUTE OF	AM 265C
023D C 0C0C	CC 0	*RUNTIME	AM 2655
023E C 0C0C	DISP DC 0	DISPLACEMENT	AM 266C
023F 00C6	TEA BSS 6	TABLE OF EA IS SIX WORDS LONG	AM 2665
0245 C 0C08	D8 CC 8	CONSTANT	AM 267C
0246 C 0C0D	D13 DC 13	CONSTANT	AM 2675
0247 C 0038	AEXIT DC \$EXIT	CONSTANT	AM 2680
0248 C 00F0	HFO DC /F0	CONSTANT	AM 2685
0249 C 0C8C	HC080 DC /0080	CONSTANT	AM 269C
024A C 0300	HO300 DC /0300	CONSTANT	AM 2695
024B C 0400	HO400 DC /0400	CONSTANT	AM 2700
024C C 03FF	HO3FF DC /03FF	CONSTANT	AM 2705
024D C 0000	LCMSK DC *-*	LENGTH OF CORE MASK	AM 271C
	*****		AM 2715
	* LEVEL 5 INTERRUPT ENTRY POINT *		AM 2720
	*****		AM 2725
	* THIS POINT IS ENTERED AFTER THE EXECUTION OF *		AM 273C
	* EACH MACHINE LANGUAGE INSTRUCTION IN THE USER *		AM 2735
	* WRITTEN PROGRAM AND USER CALLED SUBPROGRAMS. *		AM 2740
	* INTERRUPT RUN MODE, OPERATING ON LEVEL 5, IS *		AM 2745
	* USED TO IMPLEMENT THIS FUNCTION. (SEE IBM *		AM 275C
	* 1130 FUNCTIONAL CHARACTERISTICS MANUAL FOR *		AM 276C
	* FURTHER INFORMATION.) *		AM 2765
	*****		AM 277C
	* THE ACCUMULATOR, EXTENSION, CARRY, OVERFLOW, *		AM 2775
	* INDEX 1, AND INDEX 2 ARE SAVED SO THAT THEY *		AM 278C
	* MAY BE RESTORED BEFORE EXECUTION OF THE USER'S *		AM 2785
	* NEXT INSTRUCTION BY THE ROUTINE RESTC. THE *		AM 279C
	* NEXT INSTRUCTION (INST) AND ITS ADDRESS (ADDR) *		AM 2795
	* ARE LOADED, WITH THE OLD VALUES BEING STORED *		AM 280C
	* INTO LAST INSTRUCTION (LINST) AND ITS ADDRESS *		AM 2805
	* (LADDR). *		AM 281C
	*****		AM 2815
	ENT5 DC *-*	LEVEL 5 ENTRY POINT	AM 2820
024E C 0000	STX 1 SAVX1+1	SAVE INDEX 1	AM 2825
024F C 69DD	STX 2 SAVX2+1	SAVE INDEX 2	AM 283C
0250 C 6ADE	STD SAVE1	SAVE ACC AND EXT	AM 2835
0251 C D8E2	STS SAVCO	SAVE CARRY AND OVERFLOW	AM 284C
0252 C 28DE	LD ADDR	LOAD LAST ADDRESS	AM 2845
0253 C C06D	STO LADDR	STORE AS LAST ADDRESS	AM 285C
0254 C DCA8	LDD INST	LOAD DOUBLE LAST INSTRUCTION	AM 2855
0255 C C86C			AM 2860

PAGE 11

C256 C	D8A7	STD	LINST	STORE DCUBLE AS LAST INSTRUCT	AM 2865
C257 C	CCF6	LC	ENT5	LOAD ADDRESS CF INSTRUCTION	AM 2870
C258 C	ECF4	AND	LCMSK	'CIVICE' BY LENGTH CF CCRE	AM 2875
C259 C	DC67	STC	ADDR	STORE AS ADDRESS	AM 2880
C25A CC	D400002	STO	L 2	STORE INTC INDEX 2	AM 2885
C25C C	C200	LC	2 C	LCAC INSTRUCTION	AM 2890
C25D C	DC64	STO	INST	STORE AS INSTRUCTION	AM 2895
C25E C	C201	LC	2 1	LCAC SECCND WCRD OF INSTR	AM 2900
C25F C	DC63	STO	INST+1	STORE AS INSTRUCTION (LCWER)	AM 2905
*****					AM 2910
* TEST ADDRESS USING TABLE					AM 2915
*****					AM 2920
* THE INDICATOR TELLING IF THE INSTRUCTION IS A					AM 2925
* WAIT IS INITIALIZED TO ZERC (WAITC). A LCCP					AM 2930
* IS THEN PREFORMED TO DETERMINE IN WHAT PARTIT-					AM 2935
* ION OF CORE THE ADDRESS OF THE INSTRUCTION IS					AM 2940
* LOCATED. THE INDEX OF THIS LCOP IS USED TO					AM 2945
* BRANCH TO THE PROPER PCINT FOR TESTING OF THE					AM 2950
* ADDRESS.					AM 2955
*****					AM 2960
* SLA 16 ENTER ACC WITH ZERO					AM 2965
0260 C	101C	STO	WAITC	INITIALIZE WAITC TO ZERC	AM 2970
C261 C	DC96	LD	ADDR	LCAC ADDRESS	AM 2975
C262 C	C05E	LDX	2 10	ENTER INDEX 2 WITH 10	AM 2980
C263 C	62CA	BACK S	L2 TBS	SUBTRACT ENTRY IN TABLE	AM 2985
C264 C1	9600114	BSC	L CUT,Z+	GO TO OUT ON MINUS	AM 2990
C266 C1	4C28C26A	MDX	2 -1	MODIFY XR2 BY -1,SKIP IF ZERO	AM 2995
C268 0	72FF	MDX	BACK	GO TO BACK	AM 3000
C269 C	7CFA	CUT	STX 2 SADR5	SAVE XR2 IN SADR5	AM 3005
C26A C	6A54	BSC	12 TADDR	GO TO THROUGH TABLE OF ADDR	AM 3010
C26B C1	4E8C0CF4	*****			AM 3015
* ADDRESS WITHIN RESICENT MONITOR					AM 3020
*****					AM 3025
* IF THE ADDRESS IS WITHIN THE RESICENT MONITOR,					AM 3030
* THIS ROUTINE IS ENTERED. IF THE ADDRESS OF					AM 3035
* THE INSTRUCTION (ADDR) IS THE CALL EXIT ENTRY					AM 3040
* TO THE MONITOR, INDEX 2 SET TO INDICATE A					AM 3045
* NORMAL EXIT. IN EITHER CASE THE PROGRAM IS					AM 3050
* NOT ALLOWED TO CONTINUE EXECUTING. THIS IS					AM 3055
* ACCOMPLISHED BY GOING TO CGA.					AM 3060
*****					AM 3065
026D C	CC53	N402	LD ADDR	LOAD ACC WITH ADDRESS	AM 3070
026E C	F0C8	ECR	AEXIT	CCMPARE WITH EXIT ENT PCINT	AM 3075
026F C1	4C2003AC	BSC	L CGA,Z	GO TO CGA IF NOT ZERO	AM 3080
0271 C	6220	LDX	2 /20	ENTER XR2 WITH /20 AS	AM 3085
*INDICATOR CF NCRMAL EXIT					AM 3090
0272 C	704A	MDX	CG	GO TO CCMPUTE GRACE	AM 3095
*****					AM 3100
* ADDRESS WITHIN MAINLINE					AM 3105
*****					AM 3110
* IF THE ADDRESS IS WITHIN THE MAINLINE PROGRAM,					AM 3115
* THE MON INDICATOR IS SET TO ZERO TO INDICATE					AM 3120
* THAT THE PROGRAM IS WITHIN THE MAINLINE.					AM 3125
*****					AM 3130
0273 C	1010	N410	SLA 16	ENTER ACC WITH ZERO	AM 3135
0274 C	DC4F	STO	MON	STORE ZERO INTO MON INDICATOR	AM 3140
*****					AM 3145

PAGE 12

		* MON = 1 WHILE IN MONITOR	AM 3155
		* MON = 0 WHILE IN MAINLINE	AM 3160
		* MCN = -1 WHILE IN SUBROUTINES	AM 3165
0275 0 7C03	MDX N106	GO TO N106	AM 3170
		*****	AM 3175
		* ADDRESS WITHIN SUBROUTINES OR LIBF TV	AM 3180
		*****	AM 3185
		* IF THE PROGRAM IS WITHIN THE SUBROUTINE AREA	AM 3190
		* OR THE LIBF TRANSFER VECTOR, THE MCN INDICATOR	AM 3195
		* IS TESTED TO DETERMINE IF THE PROGRAM IS	AM 3200
		* VALIDLY WITHIN THESE AREAS (IE. IT MUST BE	AM 3205
		* EQUAL TO MINUS ONE).	AM 3210
		*****	AM 3215
0276 0 C04D	N414 LD MON	LOAD ACC WITH MON INDICATOR	AM 3220
0277 01 4C1003AC	BSC L CGA,-	GO TO CGA ON NOT MINUS	AM 3225
		*****	AM 3230
		* FORMS EFFECTIVE ADDRESS	AM 3235
		*****	AM 3240
		* AFTER IT HAS BEEN DETERMINED THAT THE	AM 3245
		* INSTRUCTION IS IN A VALID PARTITION OF CORE,	AM 3250
		* THE TESTING CONTINUES WITH THIS ROUTINE TO	AM 3255
		* DETERMINE IF THE EFFECTIVE ADDRESS IS IN A	AM 3260
		* VALID PARTITION OF CORE. CONSIDERATION IS	AM 3265
		* MADE OF THE TYPE INSTRUCTION INVOLVED, THE	AM 3270
		* PARTITION IN WHICH IT IS LOCATED, AND WHETHER	AM 3275
		* OR NOT IT WILL ALTER CORE IF EXECUTED. (IF	AM 3280
		* IT WILL ALTER CORE, THE STORE INDICATOR IS SET	AM 3285
		* TO ONE. IF NOT, IT IS SET TO ZERO.)	AM 3290
		*****	AM 3295
		* EFFECTIVE ADDRESSES ARE CALCULATED FOR EACH	AM 3300
		* OF THE SIX POSSIBLE TYPES OF ADDRESSING --	AM 3305
		* SHORT, SHORT INDEXED, LONG, LONG INDEXED,	AM 3310
		* INDIRECT, AND INDIRECT INDEXED.	AM 3315
		*****	AM 3320
0279 01 6680022F	N106 LDX I2	SAVX2+1 RESTORE INDEX 2	AM 3325
027B 0 C046	LD	INST LOAD INSTRUCTION FOR TESTING	AM 3330
027C 0 E0CF	AND	H03FF REMOVE ALL BUT TAG AND DISP.	AM 3335
027D 0 1888	SRT	8 MOVE -- TAG-LCW ACC, DISP-EXT	AM 3340
027E 0 D005	STO	INST+1 STORE AS ADDRESS OF INSTR.	AM 3345
027F 0 D00D	STO	INST+1 STORE AS ADDRESS OF INSTR.	AM 3350
0280 0 18D0	RTE	16 MOVE EXT (DISP) TO ACC	AM 3355
0281 0 1888	SRT	8 EXTEND SIGN OF DISPLACEMENT	AM 3360
0282 0 D0B8	STO	DISP STORE AS THE DISPLACEMENT	AM 3365
0283 00 84000000	INS1 A L	*-* ADD THE INDEX REGISTER	AM 3370
0285 0 DCBA	STO	TEA+1 STORE IN TABLE EFFECTIVE ACCR	AM 3375
0286 0 CCB7	LD	DISP LOAD THE DISPLACEMENT	AM 3380
0287 0 8C39	A	ADDR ADD THE ADDRESS OF INST	AM 3385
0288 0 803E	A	D1 ADD ONE BECAUSE IAR=1+ACCR	AM 3390
0289 0 DCB5	STO	TEA+0 STORE IN TABLE EFFECTIVE ACCR	AM 3395
028A 0 C038	LD	INST+1 LOAD SECOND WORD OF INST	AM 3400
028B 0 D0B5	STO	TEA+2 STORE IN TABLE EFFECTIVE ACCR	AM 3405
028C 00 84000000	INS2 A L	*-* ADD THE INDEX REGISTER	AM 3410
028E 0 DCB3	STO	TEA+3 STORE IN TABLE EFFECTIVE ACCR	AM 3415
028F 01 C4800241	LD I	TEA+2 LOAD INDIRECT FROM AN EA	AM 3420
0291 0 DCB1	STO	TEA+4 STORE IN TABLE EFFECTIVE ACCR	AM 3425
0292 01 C4800242	LD I	TEA+3 LOAD INDIRECT FROM AN EA	AM 3430

PAGE 13

C294 C DCAF

C295 C 610C
C296 C CC2B
C297 C ECB2
C298 C 4820
C299 C 611C
C29A C CC27
C29B C ECAF
C29C C1 4C1802A3
C29E C 7110
C29F C CC22
C2A0 C ECA8
C2A1 C 482C
C2A2 C 71F8
C2A3 C CC1E
C2A4 C 180B
C2A5 C 10C1
C2A6 C0 D4C00002
C2A8 C1 CE000084
C2AA C 19C0
C2AB C 180C
C2AC C 9C1B
C2AD C1 4C2802C9
C2AF C 18D1
C2B0 C 180F
C2B1 C DC13

C2B2 C 18C3
C2B3 C 18CD
C2B4 C0 D4C00C01
C2B6 C1 C500C23F
C2B8 C EC94
C2B9 C D0CC
C2BA C 4C24
C2BB C1 4C000200
C2BD C1 4C0003AC

C2BF C 0000
C2C0 C 0C00
C2C1 C 0C00
C2C2 C 0CC2
C2C4 C 0CC0

```

      STO      TEA+5      STORE IN TABLE EFFECTIVE ADDR      AM 3445
*****
      * TEST INSTRUCTION *
*****
      * THE INSTRUCTION IS TESTED TO DETERMINE WHICH *
      * OF THE ENTRIES IN THE EFFECTIVE ADDRESS TABLE *
      * IS IN FACT THE EFFECTIVE ADDRESS. A BRANCH *
      * IS THEN MADE TO THE ROUTINE WHICH TESTS THE *
      * EFFECTIVE ADDRESS BY DETERMINING WHICH *
      * PARTITION OF CORE IT IS IN. *
*****
      LDX      1 0      ENTER INDEX 1 WITH ZERO      AM 3500
      LD       INST     LOAD INSTRUCTION FOR TESTING      AM 3505
      AND      H0300     REMOVE ALL BUT TAG BITS      AM 3510
      BSC      Z         SKIP CN ZERO      AM 3515
      LDX      1 32-4    ENTER INDEX 1 WITH 28      AM 3520
      LD       INST     LOAD INSTRUCTION FOR TESTING      AM 3525
      AND      H0400     REMOVE ALL BUT FORMAT BIT      AM 3530
      BSC      L SHORT,+ GO TO SHORT CN ZERO      AM 3535
      MDX      1 16      MODIFY INDEX 1 BY 16      AM 3540
      LD       INST     LOAD INSTRUCTION FOR TESTING      AM 3545
      AND      H0080     REMOVE ALL BUT INDIRECT BIT      AM 3550
      BSC      Z         SKIP IF ACC ZERO      AM 3555
      MDX      1 -8      MODIFY INDEX 1 BY -8      AM 3560
      SHORT LD      INST     LOAD INSTRUCTION FOR TESTING      AM 3565
      SRA      11        REMOVE ALL BUT OP CODE      AM 3570
      SLA      1         MULTIPLY BY TWO      AM 3575
      STC      L 2        STORE ACC INTO INDEX 2      AM 3580
      LDD      L2 TINST   LOAD DOUBLE FROM TABLE INST      AM 3585
      RTE      1 0        ROTATE RIGHT ACC USING XR1      AM 3590
      SRA      12        REMOVE ALL BUT HEX DIGIT      AM 3595
      S        D4         SUBTRACT FOUR      AM 3600
      BSC      L SP,+Z    GO TO SP ON MINUS      AM 3605
      RTE      17        PLACE STORE BIT INTO HIGH ACC      AM 3610
      SRA      15        MOVE INTO LOW ACC      AM 3615
      STO      STORE     STORE AS STORE INDICATOR      AM 3620
      *
      * STORE INDICATOR = 0 IF LOAD      AM 3625
      * STORE INDICATOR = 1 IF STORE      AM 3630
      RTE      3         MOVE 3 BITS OF EXT INTO ACC      AM 3635
      SRA      16-3      RIGHT JUSTIFY THE THREE BITS      AM 3640
      STO      L 1        STORE ACC INTO INDEX 1      AM 3645
      LD       L1 TEA     LOAD FROM TABLE OF EA      AM 3650
      TEST     AND LCMSK   'DIVIDE' BY LENGTH OF CORE      AM 3655
      STO      EA         STORE AS EFFECTIVE ADDRESS      AM 3660
      BSI      TSTE      RETURN BRANCH TO TEST EA      AM 3665
      XEQN     BSC L XEQ   GO TO XEQ ACTUAL      AM 3670
      CG       BSC L CGA   GO TO CG ACTUAL      AM 3675
*****
      * CONSTANTS FOR GENERAL USE *
*****
      SADR5 DC      *-   LOCATION TO SAVE ADDR INDIC.      AM 3685
      SEAS  DC      *-   LOCATION TO SAVE EA INDICATOR      AM 3690
      ACDR  DC      0     ADDRESS OF INSTRUCTION NEXT      AM 3700
      INST BSS E 2     NEXT INSTRUCTION      AM 3710
      MCN   DC      0     MON INDICATOR      AM 3715
      * MON = 1 WHILE IN MONITOR      AM 3720
      * MON = 0 WHILE IN MAINLINE      AM 3725
      * MON = -1 WHILE IN SUBROUTINES      AM 3730

```


PAGE 14

02C5 0 0000

02C6 0 0000

02C7 0 0001

02C8 0 0004

02C9 0 621C

02CA 00 D4000001

02CC 01 4D8002D2

02CE 1 0200

02CF 1 03AC

02D0 1 02D2

02D1 1 02D6

02D2 01 74FF01F8

02D2

02D4 0 1000

02D5 0 70E5

02D6 0 6100

02D7 01 4400036F

02D9 0 FFFF

02DA 0 7400

02DB 0 6101

02DC 0 69E8

02DD 0 C0E5

02DE 0 7CD9

02DF 0 0000

02E0 0 620A

02E1 01 96000114

02E3 01 4C2802E7

02E5 0 72FF

02E6 0 70FA

02E7 0 720B

02E8 0 6AD7

02E9 01 4E8000F4

```

STORE DC      0      STORE INDICATOR
*              STORE INDICATOR = 0 IF LCAD
*              STORE INDICATOR = 1 IF STORE
EA      DC      0      EFFECTIVE ADDRESS STORAGE
D1      DC      1      CONSTANT
D4      DC      4      CONSTANT
*****
*      TEST FOR SPECIAL CASES
*****
*      THIS ROUTINE IS ENTERED WHEN IT IS DESIRED TO
*      TEST THE SPECIAL CASES WHERE THE INSTRUCTION
*      HAS NO EFFECTIVE ADDRESS.  THESE CASES ARE
*      CLASSIFIED FOUR WAYS -- INSTRUCTIONS THAT CAN
*      NOT BE ALLOWED TO EXECUTE, INSTRUCTIONS THAT
*      ARE ALWAYS ALLOWED TO EXECUTE, WAIT INSTR-
*      UCTIONS, AND THE MDX INSTRUCTION.
*****
SP      LDX      2 /1C      ENTER XR2 WITH /1C AS INDIC.
          STO      L 1      STORE ACC INTO XR1
          BSC      11 TSPR      BRANCH THROUGH TSPR USING XR1
          DC      XEQ      GO TO XEQ
          DC      CGA      GO TO CGA
          DC      WAITS      GO TO WAITS
          DC      MDXLO      GO TO MDXLO
          WAITS MDX L      WAITC,-1 DECREMENT WAITC TO NEGATIVE
          TSPR EQU      WAITS
          NOP
          MDX      XEQN      GO TO XEQ
          MDXLO LDX      1 0      ENTER INDEX 1 WITH ZERO
          BSI      L BITS      RETURN BRANCH TO BITS
          DC      /FFFF      TEST FOR MDX INSTR. WITH
          DC      /7400      LONG FORMAT, NO INDEX, AND
          *              *ZERO DISPLACEMENT
          LDX      1 1      ENTER XR1 WITH 1 IF FALSE
          STX      1 STORE      STORE INDEX 1 AS STORE INC.
          LD      INST+1      LOAD 2ND HALF OF INSTRUCTION
          MDX      TEST      GO TO TEST
*****
*      TEST EFFECTIVE ADDRESS USING TABLE
*****
*      THE EFFECTIVE ADDRESS IS TESTED BY DETERMINING
*      IN WHICH PARTITION OF CORE IT LIES.
*****
TSTEA DC      *-*      ENTRY PCINT FOR TEST EA
          LDX      2 10      ENTER INDEX 2 WITH 10
          BACK1 S      L2 TBS      SUBTRACT ENTRY IN TABLE
          BSC      L OUT1,Z+      GO TO OUT1 ON MINUS
          MDX      2 -1      MODIFY XR2 BY -1, SKIP IF ZERO
          MDX      BACK1      GO TO BACK1
          OUT1 MDX      2 11      MODIFY XR2 BY 11 IN ORDER TO
          *              *USE THE LOWER HALF OF TACCR
          STX      2 SEAS      SAVE XR2 IN SEAS
          BSC      12 TACCR      GO TO THROUGH TABLE OF ADDR
*****
*      EFFECTIVE ADDRESS IN FIRST FOUR WORDS
*****

```

AM 3735
AM 3740
AM 3745
AM 3750
AM 3755
AM 3760
AM 3765
AM 3770
AM 3775
AM 3780
AM 3785
AM 3790
AM 3795
AM 3800
AM 3805
AM 3810
AM 3815
AM 3820
AM 3825
AM 3830
AM 3835
AM 3840
AM 3845
AM 3850
AM 3855
AM 3860
AM 3865
AM 3870
AM 3875
AM 3880
AM 3885
AM 3890
AM 3895
AM 3900
AM 3905
AM 3910
AM 3915
AM 3920
AM 3925
AM 3930
AM 3935
AM 3940
AM 3945
AM 3950
AM 3955
AM 3960
AM 3965
AM 3970
AM 3975
AM 3980
AM 3985
AM 3990
AM 3995
AM 4000
AM 4005
AM 4010
AM 4015
AM 4020

C2EB 0 CODA
C2EC 01 4C200365

C2EE 0 C0D7
C2EF 0 FC77

C2F0 01 4C180365
C2F2 0 4C7C
C2F3 0 FC00
C2F4 0 440C
C2F5 0 7C29
C2F6 0 6106
C2F7 0 C0CE
C2F8 01 F50C00CE

C2FA 01 4C180313
C2FC 0 71FF
C2FD 0 7CF9
C2FE 0 CCC7
C2FF 0 F01E
C300 01 4C2003AC
C302 01 658002C1
C304 0 C103
C305 0 D0C7
C306 0 C1C4
C307 0 D0C6
C308 01 4400022B
C3CA

C30F 01 7405024E
C311 01 4C00024F
C313 01 4400022B
C315 01 448002C6
C317 01 7402024E
C319 01 4C00024F
C31C 00C2
C31E 0 0C3F

```

* IF THE EFFECTIVE ADDRESS IS EQUAL TO ZERO, * AM 4025
* IT IS TESTED AS PART OF THE MONITOR. IF THE * AM 4030
* EFFECTIVE ADDRESS IS WITHIN THE INDEX * AM 4035
* REGISTERS (LOCATIONS ONE, TWO, AND THREE IN * AM 4040
* CCRE), A BRANCH IS MADE TO XEQ IN ORDER TO * AM 4045
* RETURN TO THE CALLING PROGRAM. * AM 4050
***** AM 4055
N500 LD EA LOAD EFFECTIVE ADDRESS AM 4060
      BSC L XXX,Z GO TO XEQ ON NOT ZERO AM 4065
***** AM 4070
* EFFECTIVE ADDRESS IN RESIDENT MONITOR * AM 4075
***** AM 4080
* IF THE EFFECTIVE ADDRESS IS WITHIN THE MONITOR * AM 4085
* AREA, THE PROGRAM IS NOT ALLOWED TO ENTER THE * AM 4090
* MONITOR. IF THE ATTEMPTED ENTRY POINT IS AN * AM 4095
* I/O ERROR TRAP IN 'LIST', OR IS THE DUMP * AM 4100
* ENTRY, THEN THE ENTRY IS MADE FROM THIS AMS * AM 4105
* ROUTINE RATHER THAN THE CALLING PROGRAM. * AM 4110
***** AM 4115
N501 LD EA LOAD EFFECTIVE ADDRESS AM 4120
      ECR IOCT COMPARE WITH I/O OFF INDIC- AM 4125
      *ATOR ADDRESS AM 4130
      BSC L XXX,+ GO TO XXX ON ZERO AM 4135
      BSI BITS RETURN BRANCH TO BITS AM 4140
      DC /FCC0 CHECK FOR LONG AM 4145
      DC /4400 BSI INSTRUCTION (0100C1). AM 4150
      MDX N503 GO TO N503 IF TEST FALSE AM 4155
      LCX 1 6 ENTER INDEX 1 WITH 6 AM 4160
RT LD EA LOAD EA FOR COMPARISON AM 4165
   EOR L1 LIST COMPARE AN ENTRY ADDRESS AM 4170
   * TO THE CORE RESIDENT MONITOR AM 4175
      BSC L X,-+ GO TO X IF ACC ZERO AM 4180
      MDX 1 -1 MODIFY XR1 BY -1, SKIP IF ZERO AM 4185
      MDX RT GO TO RT AM 4190
      LD EA LOAD EFFECTIVE ADDRESS AM 4195
      EOR ADMP CCMPARE WITH DUMP ENTRY ADDR. AM 4200
      BSC L CGA,Z GO TO CGA IF NOT ZERO AM 4205
      LDX I1 ADDR LOAD XR1 WITH ADDRESS AM 4210
      LD 1 3 LOAD BEGINING ADDRESS AM 4215
      STO D+3 STORE INTO PCMP STATEMENT AM 4220
      LD 1 4 LOAD END ADDRESS AM 4225
      STO D+4 STORE INTO PCMP STATEMENT AM 4230
      BSI L RESTO RESTORE ACC,EXT,XR1,XR2,C,O. AM 4235
D PDMP *-*,*-* DUMP CORE AS SPECIFIED BY AM 4240
* *THE CALLING PROGRAM. AM 4245
      MDX L ENT5,5 MODIFY RETURN ADDRESS BY 5 AM 4250
      BSC L ENT5+1 RETURN TO TEST NEXT INSTR. AM 4255
X BSI L RESTO RESTORE ACC,EXT,XR1,XR2,C,C. AM 4260
  BSI I EA BRANCH (BSI) TO EFF. ADDR. AM 4265
  MDX L ENT5,2 MODIFY RETURN ADDRESS BY 2 AM 4270
  BSC L ENT5+1 RETURN TO TEST NEXT INSTR. AM 4275
IOAR3 BSS E 2 TOP OF DISK BUFFER 3 AM 4280
ACMP DC $DUMP DUMP ENTRY POINT AM 4285
***** AM 4290
* EFFECTIVE ADDRESS IN AMS OR ILS AREA * AM 4295
***** AM 4300
* IF THE INSTRUCTION IS OF A TYPE THAT ALTERS * AM 4305
* CCRE (IE. THE STORE INDICATOR IS EQUAL TO ONE) * AM 4310

```

031F 0 COA5
0320 01 4C2003AC

0322 0 C049
0323 01 4C180334
0325 00 D4000001
0327 0 C099
0328 01 F5000002
032A 01 4C180365
032C 0 71FF
032D 0 70F9
032E 0 C030
032F 0 903A
0330 01 4C280334
0332 0 621A
0333 0 7089
0334 01 7401036C
0336 01 6580036C
0338 0 C088
0339 01 D5000C02
033B 0 7029

033C 0 C087
033D 01 4C280365

033F 0 402F
0340 0 FF80
0341 0 4480

```

* THEN IT IS NOT ALLOWED TO EXECUTE. IF NOT, IT *
* IS TREATED AS IF THE EA IS WITHIN UNUSED CCRE. *
*****
N503 LD STORE LCAC STCRE INDICATOR
      BSC L CGA,Z GO TO CCMPUTE GRADE IF NCT C
* STCRE INDICATOR = 0 IF LCAC
* STCRE INDICATOR = 1 IF STCRE
*****
* EFFECTIVE ADDRESS IN UNUSED CCRE (GARBAGE) *
*****
* IF THE EFFECTIVE ADDRESS (EA) IS WITHIN UNUSED *
* CORE (GARBAGE), THEN THE ADDRESS IS *
* RECORDED IN THE GARBAGE TABLE. IF THAT *
* ADDRESS IS ALREADY IN THE GARBAGE TABLE, NO *
* NEW ENTRY IS MADE IN THE TABLE. IF THE TABLE *
* BECOMES FULL, THE PROGRAM IS ABORTED AND A *
* SPECIAL ERROR MESSAGE IS PRINTED BY THE OUTPUT *
* PROGRAM. *
*****
N504 LD GCTR LOAD GARBAGE COUNTER
      BSC L PP,-+ GO TO PP IF ZERO
      STO L 1 STORE INTO INDEX 1
BCK LD ADDR LOAD ACC WITH ADDRESS
     EOR L1 GTBL-1 CCMPARE WITH GARBAGE TABLE
      BSC L XXX,-+ GO TO XEQ CN ZERO
      MDX 1 -1 MODIFY XR1 BY -1,SKIP IF ZERO
      MDX BCK GO TO BCK IF NO SKIP
PAST LD GCTR LOAD GARBAGE COUNTER
      S D64 SUBTRACT 64 (LENGTH OF GTBL)
      BSC L PP,Z+ GO TO PP IF NEGATIVE
      LDX 2 /1A ENTER XR2 WITH /1A AS INDIC.
      MDX CG GO TO CCMPUTE GRADE
PP MDX L GCTR,1 INCRIMENT GARBAGE COUNTER
   LDX I1 GCTR LOAD XR1 WITH GARBAGE COUNTER
   LD ADDR LOAD ACC WITH ADDRESS
   STO L1 GTBL-1 STORE INTO GARBAGE TABLE
   MDX XXX GO TO XEQ
*****
* EFFECTIVE ADDRESS IN SUBROUTINES *
*****
* IF THE EFFECTIVE ADDRESS IS WITHIN THE SUBROUT- *
* INES, IT IS PERMITTED TO EXECUTE IF THE *
* INSTRUCTION IS WITHIN THE SUBROUTINES, OR IF *
* THE INSTRUCTION IS A VALID CALL ENTRY TO THE *
* SUBROUTINE AREA THROUGH THE CALL TRANSFER *
* VECTOR. IN THIS LAST CASE, THE MCN INDICATOR *
* IS SET TO INDICATE THAT THE PROGRAM IS VALIDLY *
* WITHIN THE SUBROUTINE AREA. IF BOTH THESE *
* TESTS FAIL, THE EFFECTIVE ADDRESS IS TREATED *
* AS IF WITHIN AMS OR ILS AREA (IE. IT IS WITHIN *
* AN AREA THAT MUST NOT BE ALTERED.) *
*****
N506 LD MON LOAD MON INDICATOR
      BSC L XXX,Z+ GO TO XEQ IF PROGRAM IS IN
* SUBROUTINE AREA
      BSI BITS RETURN BRANCH TO BITS ROUTINE
      DC /FF80 CHECK FOR BSI IO
      DC /4480 INSTRUCTION (C100C1001).

```

AM 4315
AM 4320
AM 4325
AM 4330
AM 4335
AM 4340
AM 4345
AM 4350
AM 4355
AM 4360
AM 4365
AM 4370
AM 4375
AM 4380
AM 4385
AM 4390
AM 4395
AM 4400
AM 4405
AM 4410
AM 4415
AM 4420
AM 4425
AM 4430
AM 4435
AM 4440
AM 4445
AM 4450
AM 4455
AM 4460
AM 4465
AM 4470
AM 4475
AM 4480
AM 4485
AM 4490
AM 4495
AM 4500
AM 4505
AM 4510
AM 4515
AM 4520
AM 4525
AM 4530
AM 4535
AM 4540
AM 4545
AM 4550
AM 4555
AM 4560
AM 4565
AM 4570
AM 4575
AM 4580
AM 4585
AM 4590
AM 4595
AM 4600

PAGE 17

C342 C	7C0C	MDX	N503	GC TO N503 IF CHECK FALSE	AM 4605
C343 C1	C40002C3	LD	L	INST+1 PUT ADDRESS PORTION INTO ACC	AM 4610
C345 C	9C27	S		BCALL SUBTRACT ADDR BEG. OF CALL	AM 4615
C346 C1	4C2803AC	BSC	L	CGA,+Z GO TO CGA ON MINUS	AM 4620
C348 C1	94000115	S	L	LCLTV SUBTRACT LENGTH OF CALL	AM 4625
C34A C1	4C1003AC	BSC	L	CGA,- GC TO CGA ON NOT MINUS	AM 4630
C34C C	7C11	MDX	N508	GC TO N508	AM 4635
				*****	AM 4640
				* EFFECTIVE ADDRESS IN LIBF T V *	AM 4645
				*****	AM 4650
				* IF THE EFFECTIVE ADDRESS IS WITHIN THE *	AM 4655
				* LIBF TRANSFER VECTOR, THE INSTRUCTION, INDEX *	AM 4660
				* THREE, AND THE EFFECTIVE ADDRESS ARE TESTED *	AM 4665
				* TO DETERMINE IF IT IS A PROPER ENTRY INTO THE *	AM 4670
				* LIBF TV. IF THE TEST FAILS, IT IS TREATED AS *	AM 4675
				* IF THE EA WAS WITHIN THE CALL TRANSFER VECTOR. *	AM 4680
				* IF THE TEST IS SUCCESSFUL, THEN THE MON *	AM 4685
				* INDICATOR IS SET TO INDICATE THAT IT IS VALID *	AM 4690
				* FOR THE PROGRAM TO BE WITHIN THE SUBROUTINE *	AM 4695
				* OR LIBF TV AREAS. *	AM 4700
				*****	AM 4705
C34D C	4C21	N507	BSI	BITS RETURN BRANCH TO BITS	AM 4710
C34E C	FF00	DC		/FF00 CHECK FOR SHORT BSI INSTR.	AM 4715
C34F C	430C	DC		/4300 WITH XR3 (C1000011).	AM 4720
C350 C	7C11	MDX	N510	GO TO N510 IF CHECK FALSE	AM 4725
C351 C0	C4000C03	LC	L	3 LOAD INDEX 3	AM 4730
C353 C	FC74	EOR		SPXR3+1 COMPARE WITH PROPER VALUE XR3	AM 4735
C354 C1	4C2003AC	BSC	L	CGA,Z GO TO CG ACTUAL IF NOT ZERO	AM 4740
C356 C	CC17	LD		BLBTV LOAD LOW END ADDR LIBF TV	AM 4745
C357 C1	940002C6	S	L	EA SUBTRACT EFFECTIVE ADDRESS	AM 4750
C359 C	1890	SRT		16 SHIFT INTO A TWO WORD OPERAND	AM 4755
C35A C	A8CE	D		D3 DIVIDE BY THREE	AM 4760
C35B C	18C0	RTE		16 PLACE EXT INTO ACC	AM 4765
C35C C1	4C2003AC	BSC	L	CGA,Z GO TO COMPUTE GR IF NOT ZERO	AM 4770
C35E C	CC09	N508	LD	DM1 LOAD ACC WITH MINUS ONE	AM 4775
C35F C1	D40002C4	STO	L	MON STORE INTO MON INDICATOR	AM 4780
				MON = 1 WHILE IN MONITOR	AM 4785
				MON = 0 WHILE IN MAINLINE	AM 4790
				MON = -1 WHILE IN SUBROUTINES	AM 4795
				*****	AM 4800
C361 C	7C03	MDX	XXX	GO TO XEQ	AM 4805
				*****	AM 4810
				* EFFECTIVE ADDRESS IN CALL T V *	AM 4815
				*****	AM 4820
				* IF THE INSTRUCTION IS OF A TYPE THAT ALTERS *	AM 4825
				* CORE, IT WILL NOT BE PERMITTED TO EXECUTE. *	AM 4830
				* IF IT IS NOT OF A TYPE THAT ALTERS CORE, IT *	AM 4835
				* WILL BE PERMITTED TO EXECUTE, WITHOUT AN ENTRY *	AM 4840
				* IN THE GARBAGE TABLE. *	AM 4845
				*****	AM 4850
C362 C1	740002C5	N510	MDX	L STORE,0 SKIP IF STORE INDIC. IS ZERO	AM 4855
C364 C	7047	MDX		CGA GO TO CGA IF NOT ZERO	AM 4860
				STORE INDICATOR = 0 IF LOAD	AM 4865
				STORE INDICATOR = 1 IF STORE	AM 4870
C365 C1	4C8002DF	XXX	BSC	I TSTEA EXIT FROM TEST EA ROUTINE	AM 4875
				*****	AM 4880
				* CONSTANTS FOR USE BY LOWER HALF *	AM 4885
				*****	AM 4890
C367 C	0C32	IOCT	DC	\$IOCT	

PAGE 18

0368 0	FFFF	DM1 DC	-1	CCONSTANT	AM 4895
0369 0	00C3	D3 DC	3	CCONSTANT	AM 4900
036A 0	0040	D64 DC	64	CCONSTANT	AM 4905
036B 0	0032	H32 DC	/32	CONSTANT	AM 4910
036C 0	0000	GCTR DC	0	GARBAGE COUNTER	AM 4915
036D 0	0000	BCALL DC	*--	BEGINING CF CALL TV	AM 4920
036E 0	0000	BLBTV DC	*--	BEGINING CF LIBF T V	AM 4925
*****					AM 4930
* B I T S ROUTINE					AM 4935
*****					AM 4940
* ROUTINE TO TEST BITS OF THE INSTRUCTION.					AM 4945
* FIRST WORD OF CALLING SEQUENCE INDICATES WHICH					AM 4950
* BITS ARE TO BE TESTED, THE SECOND TELLS WHICH					AM 4955
* OF THESE MUST BE SET FOR TEST TO HOLD. EXITS					AM 4960
* AT THIRD WORD IF THE TEST DOES NOT HOLD, AT					AM 4965
* THE FOURTH WORD IF IT DOES HOLD.					AM 4970
*****					AM 4975
036F 0	0000	BITS DC	*--	ENTRY PCINT FOR BITS ROUTINE	AM 4980
0370 01	C40002C2	LD L	INST	LOAD INSTRUCTION FOR TESTING	AM 4990
0372 0	6909	STX 1	SXR1+1	SAVE INDEX 1	AM 4995
0373 01	6580036F	LDX 11	BITS	LOAD RETURN ADDR INTO XR1	AM 5000
0375 0	E100	AND X1	0	LEAVE SET ONLY BITS DESIRED	AM 5005
0376 0	F1C1	ECR X1	+1	COMPLEMENT DESIRED BITS	AM 5010
0377 0	4818	BSC	+-	SKIP ON NOT ZERO (TEST FALSE)	AM 5015
0378 0	7101	MDX 1	+1	INCREMENT XR1 IF TEST HOLDS	AM 5020
0379 0	7102	MDX 1	+2	MODIFY XR1 BY TWO	AM 5025
037A 0	6903	STX 1	RETU+1	STORE XR1 INTO ADDR OF INSTR	AM 5030
037B 00	65000000	SXR1 LDX 11	*--	RESTORE INDEX 1	AM 5035
037D 00	4C000000	RETU BSC L	*--	EXIT FROM BITS ROUTINE	AM 5040
*****					AM 5045
* IOND - ROUTINE TO WAIT FOR ALL I/C CFF					AM 5050
*****					AM 5055
037F 0	0000	IOND DC	*--	ENTRY PCINT FOR IOND	AM 5060
0380 0	4838	BSC	+-Z	SKIP	AM 5065
0381 0	3000	BACKB WAIT		WAIT FOR INTERRUPT	AM 5070
0382 0	61CA	LDX 1	10	LOAD XR1 WITH TEN	AM 5075
0383 01	0D0000C2	LCOPB X10 11	IOCCB-2	SENSE CSW FOR DISK	AM 5080
0385 0	E00E	AND	H1000	AND OUT ALL BUT BUSY BIT	AM 5085
0386 01	4C200381	BSC L	BACKB,Z	GO TO BACKB IF NOT ZERO	AM 5090
0388 0	71FE	MDX 1	-2	MODIFY XR1 BY -2, SKIP IF ZERO	AM 5095
0389 0	7CF9	MDX	LCOPB	GO TO LCOPB (IF NO SKIP)	AM 5100
038A 01	747F0392	ICH MDX L	ICH1,127	* WAIT FOR APROX.	AM 5105
038C 0	7CFD	MDX	ICH	* THREE SECONDS	AM 5110
038D 01	743F0393	MDX L	ICH2,63	*	AM 5115
038F 0	7CFA	MDX	ICH	*	AM 5120
0390 01	4C80037F	BSC I	IOND	RETURN TO CALLING PCINT	AM 5125
0392 0	0000	ICH1 DC	0	COUNTER	AM 5130
0393 0	0000	ICH2 DC	0	COUNTER	AM 5135
0394 0	1000	H1000 DC	/1000	CONSTANT	AM 5140
*****					AM 5145
* READ ROUTINE - READS CON SW. INTO ACC					AM 5150
*****					AM 5155
* READ ROUTINE - READS CON SW. INTO ACC					AM 5160
*****					AM 5165
* READ ROUTINE - READS CON SW. INTO ACC					AM 5170
*****					AM 5175
* READ ROUTINE - READS CON SW. INTO ACC					AM 5180

PAGE 19

C395 C 0000
C396 C 0803
C397 C 0004
C398 01 4C800395
C39A 0000
C39A 1 039C
C39B C 3A00
C39C C 0000

C39D C 0000
C39E C 4CF6
C39F C ECC8
C3A0 C1 4C98039D
C3A2 C 0000
C3A3 C 0003
C3A4
C3A9 C1 4C80039D
C3AB C 0002

C3AC 0000
C3AC C 4002
C3AD C 4CE7
C3AE C EC4D
C3AF C1 4C1803C4
C3B1 C0 44000002
C3B3 C1 E400024C
C3B5 C 0005
C3B6 C1 44000206
C3B8 C 1800
C3B9 C1 440002C1
C3BB C 3000
C3BC C 4000
C3BD C 3000
C3BE C 4006
C3BF C1 E4000208
C3C1 C1 4C200200
C3C3 C 4838
C3C4 C 4008

```

READ  CC      *-*      ENTRY PCINT FOR READ      AM 5185
      XIC      ICCCN    READ THE CONSOLE SWITCHES  AM 5190
      LU       RPAUS    LOAD THE NUMBER READ IN   AM 5195
      BSC I     READ    RETURN TO CALLING PCINT    AM 5200
      ICCCN BSS E 0     IOCC TO READ THE CONSOLE   AM 5205
      DC       RPAUS    ENTRY SWITCHES INTO CORE  AM 5210
      DC       /3A00    AT LOCATION RPAUS.         AM 5215
      RPAUS CC      *-*      CON SW READ INTO HERE  AM 5220
*****
*                                     * AM 5225
*                                     * AM 5230
*   DUMP -- ROUTINE TO DUMP CORE IF SW 14 SET * AM 5235
*                                     * AM 5240
*****
DUMP  CC      *-*      DUMP ENTRY PCINT           AM 5245
      BSI      READ    READ CON SW INTO ACC       AM 5250
      AND      D2      REMOVE ALL BUT BIT 14      AM 5255
      BSC I     DUMP,+  RETURN TO CALLING PCINT ON C AM 5260
      LD       BLBTV   LOAD ADDRESS BEGINING LIBF TV AM 5265
      STO      DMP+3   STORE AS PARAMETER FOR DUMP AM 5270
      DMP      PDMP -*,*-* DUMP ALL BUT UNUSED CORE AM 5275
      BSC I     DUMP    RETURN TO CALLING PCINT    AM 5280
      D2       CC      2      CONSTANT            AM 5285
*****
*                                     * AM 5290
*                                     * AM 5295
*   C G - CCMPUTE GRACE                     * AM 5300
*                                     * AM 5305
*                                     * AM 5310
*****
* THIS ROUTINE IS ENTERED WHEN IT IS DESIRED TO * AM 5315
* ABORT THE CURRENT PROGRAM AND OUTPUT THE      * AM 5320
* REASONS FOR ABORTING FURTHER EXECUTION. THIS * AM 5325
* IS ACCOMPLISHED BY PLACING INTO CORE WITHIN * AM 5330
* THIS PROGRAM ANY PARAMETERS WHICH MIGHT BE   * AM 5335
* NEEDED BY THE OUTPUT ROUTINE, WRITING THIS   * AM 5340
* PROGRAM ITSELF ON THE DISK, THEN LINKING TO * AM 5345
* THE OUTPUT ROUTINE CALLED CBUG. THIS OUTPUT * AM 5350
* ROUTINE IS RESPONSABLE FOR INTERPRETING THESE * AM 5355
* PARAMETERS ON THE DISK AND OUTPUTTING THEM IN * AM 5360
* READABLE FORM ON THE PRINCIPLE OUTPUT DEVICE. * AM 5365
*****
CGA   BSS      0      ENTRY PT FOR COMPUTE GRACE  AM 5370
      BSI      IONC    WAIT FOR ALL I/O OFF      AM 5375
      BSI      READ    REAC CON SW INTO ACC      AM 5380
      AND      H0001   REMOVE ALL BUT LOW BIT    AM 5385
      BSC L     NCWT,+  GO TO NCWT ON ZERO        AM 5390
      LD       L 2     LOAD ACC WITH INDEX 2      AM 5395
      AND      L H03FF  MAKE CP CODE ZERO         AM 5400
      STO      WAITE   STORE AS WAIT INSTRUCTION AM 5405
      LD       L EA     LOAD EFFECTIVE ADDRESS    AM 5410
      RTE      16      MOVE ACC TO EXT           AM 5415
      LD       L ADDR   LOAD ADDR OF INSTRUCTION AM 5420
      WAITE    WAIT    WAIT FOR OPERATOR         AM 5425
      BSI      DUMP    DUMP CORE IF SWITCH 14 IS ON AM 5430
      WAIT     WAIT    WAIT FOR OPERATOR         AM 5435
      BSI      READ    REAC CON SW INTO ACC      AM 5440
      AND      L D4     REMOVE ALL BUT BIT 13     AM 5445
      BSC L     XEQ,Z   GO TO XEQ IF NOT ZERO     AM 5450
      BSC      +-Z     SKIP UNCONDITIONAL         AM 5455
      NCWT     BSI      DUMP    DUMP CORE IF SWITCH 14 IS ON AM 5460

```

03C5 0 6B2F	.STX 3 XR3	STORE XR3 INTO XR3	AM 5475
03C6 0 6A2D	STX 2 XR2	STORE XR2 INTO XR2	AM 5480
03C7 00 67000C00	SPXR3 LDX L3 **	RESTORE PROPER VALUE XR3	AM 5485
03C9 0 CB7A	LDD 3 122	LOAD FIRST 2 WORDS OF FAC	AM 5490
03CA 0 D823	STD SAVF1	SAVE FOR OUTPUT ROUTINE	AM 5495
03CB 0 CB7C	LDD 3 124	LOAD SECOND 2 WORDS OF FAC	AM 5500
03CC 0 D823	STD SAVF2	SAVE FOR OUTPUT ROUTINE	AM 5505
03CD 0 CB7E	LDD 3 126	LOAD THIRD 2 WORDS OF FAC	AM 5510
03CE 0 D823	STD SAVF3	SAVE FOR OUTPUT ROUTINE	AM 5515
03CF 01 CC000238	LDD L INSCT	LOAD DOUBLE INSTRUCTION CT	AM 5520
03D1 0 A833	D D1E4	DIVIDE BY 10000	AM 5525
03D2 01 DC000238	STD L INSCT	STORE DOUBLE INSTR COUNT	AM 5530
03D4 00 C4000032	LD L \$IOCT	LOAD I/C BUSY INDICATOR	AM 5535
03D6 0 DC1F	STC SIOCT	SAVE IN SIOCT	AM 5540
03D7 0 1010	SLA 16	CLEAR ACC	AM 5545
03D8 00 D40C0032	STO L \$IOCT	CLEAR I/O BUSY INDICATOR	AM 5550
03DA 00 D4C00CEE	STO L \$DBSY	CLEAR DISK BUSY INDICATOR	AM 5555
03DC 0 6106	LDD 1 6	ENTER INDEX 1 WITH 6	AM 5560
03DD 0 C824	DLOOP LDD ICAR	LOAD SECTOR LENGTH AND ADDR.	AM 5565
03DE 0 CC1A	LD D320	LOAD 320 AS SECTOR LENGTH	AM 5570
03DF 01 DD8003FB	STD I1 DPAR-1	STORE AT TOP OF BUFFER	AM 5575
03E1 01 CC0003FA	LDD L1 DPAR-2	LOAD DISK PARAMETERS	AM 5580
03E3 00 440000F2	BSI L D2000	GO TO DISK ROUTINE	AM 5585
03E5 01 74010403	MDX L IOAR+1,1	MODIFY SECTOR ADDRESS BY ONE	AM 5590
03E7 0 71FE	MDX 1 -2	MODIFY XR1 BY -2, SKIP IF ZERO	AM 5595
03E8 0 70F4	MDX DLOCP	GO TO DLOCP ON NO SKIP	AM 5600
03E9 00 040A41E3	LINK DBUGT	CALL LINK TO DBUGT	AM 5605
03EE 0002	SAVF1 BSS E 2	LOCATION TO SAVE FAC	AM 5610
03F0 0002	SAVF2 BSS E 2	LOCATION TO SAVE FAC	AM 5615
03F2 0002	SAVF3 BSS E 2	LOCATION TO SAVE FAC	AM 5620
03F4 0 0C00	XR2 DC **	LOCATION TO SAVE GRADE INC.	AM 5625
03F5 0 0000	XR3 DC **	LOCATION TO SAVE END VALUE	AM 5630
	*	*OF INDEX 3	AM 5635
03F6 0 0000	SIOCT DC **	LOCATION TO SAVE I/C BUSY INC	AM 5640
03F7 0 EEEE	HEEEE DC /EEEE	CONSTANT	AM 5645
03F8 0 0020	H20 DC /20	CONSTANT	AM 5650
03F9 0 0140	D320 DC 320	CONSTANT	AM 5655
03FA 0 2000	H2000 DC /2000	CONSTANT	AM 5660
03FC 0000	H0001 BSS E 0	CONSTANT (ONE)	AM 5665
03FC 0000	DPAR BSS E 0	TABLE OF DISK PARAMETERS	AM 5670
03FC 0 0001	DC 1	DISK WRITE	AM 5675
03FD 1 031C	DC ICAR3	ADDRESS OF DISK BUFFER 3	AM 5680
03FE 0 0001	DC 1	DISK WRITE	AM 5685
03FF 1 01CE	DC IOAR2	ADDRESS OF DISK BUFFER 2	AM 5690
0400 0 0001	DC 1	DISK WRITE	AM 5695
0401 1 0000	DC ICAR1	ADDRESS OF DISK BUFFER 1	AM 5700
0402 31 22065109	IOAR DSA SAVGR	DISK FILE LENGTH, SECTOR	AM 5705
	*	*ADDRESS, AND AC OF SECTORS.	AM 5710
0405 0 2710	D1E4 DC 10000	CONSTANT	AM 5715
C001	'CMCN EQU 1	LENGTH OF COMMON	AM 5720
C004	'HWET EQU 4	LENGTH OF CORE IMAGE HEADER	AM 5725
C008	'TVWC EQU 8	LENGTH OF TRANSFER VECTOR	AM 5730
C009	'WCNT EQU 9	LENGTH OF CORE LOAD	AM 5735
C00A	'XR3X EQU /A	SETTING FOR INDEX 3	AM 5740
C018	'ITCK EQU 24	INTERRUPT ENTRY TO KBC/CCN PR	AM 5745
000E	\$CORE EQU /00CE	SIZE OF CORE	AM 5750
0028	\$PRET EQU /0028	PRE-CP I/O ERROR TRAP	AM 5755
C032	\$IOCT EQU /0032	I/O BUSY INDICATOR	AM 5760

PAGE 21

C038		\$EXIT EQU	/0038	ENTRY PCINT FOR EXIT	AM 5765
C03F		\$CUMP EQU	/003F	CUMP ENTRY PCINT	AM 5770
C07B		\$WRD1 EQU	/007B	LCACING ADDR CF THE CORE LCAC	AM 5775
CC81		\$PST1 EQU	/0081	POST-CP I/C ERROR TRAP L 1	AM 5780
C085		\$PST2 EQU	/0085	POST-CP I/C ERROR TRAP L 2	AM 5785
C089		\$PST3 EQU	/0089	POST-CP I/C ERROR TRAP L 3	AM 5790
CG8D		\$PST4 EQU	/008D	POST-CP I/C ERROR TRAP L 4	AM 5795
C091		\$STCP EQU	/0091	PROGRAM STCP KEY TRAP L 5	AM 5800
COEE		\$DBSY EQU	/00EE	DISK BUSY INDICATOR	AM 5805
COF2		DZ00C EQU	/00F2	DISK ROUTINE ENTRY ADDRESS	AM 5810
		*****			AM 5815
C406	0C0C	EAMS BSS E C		LAST LOCATION IN AMS PRG.	AM 5820
C406		END		END OF AM PROGRAM	AM 5825

000 OVERFLW SECTORS SPECIFIED
 000 OVERFLW SECTORS REQUIRED
 205 SYMBOLS DEFINED
 NO ERRCR(S) FLAGGED IN ABOVE ASSEMBLY

// DLP

*DELETE		UA	AM		
CART ID	0C26	DB ADDR	4E41	CB CNT	C034
*STORE	WS	UA	AM		
CART ID	0C26	DB ADDR	4EE2	CB CNT	CC34